UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
POS GRADUAÇÃO EM CIENCIA DA COMPUTAÇÃO

GERADOR PARAMETRIZAVEL DE PARTES OPERATIVAS CMOS

por

Luigi Carro

Dissertação submetida como requisito parcial
para obtenção do grau de Mestre em
Ciência da Computação

Prof. Altamiro Amadeu Suzim
Orientador

Porto Alegre, março de 1989.

CATALOGAÇÃO NA FONTE

Carro, Luigi

Gerador Parametrizável de Partes Operativas CMOS.

Porto Alegre, PGCC da UFRGS, 1989.

lv.

Diss. (mestr. ci, comp.) UFRGS-PGCC, Porto Alegre, BR-RS, 1989.

Dissertação: Microeletrônica: PAC de Circuitos Integrados: Compilador de Silício: Projeto VLSI: Circuitos Complexos.

It was not growing up that slowly applied brakes to learning (Mentats were taught) but an accumulation of "things I know".

Duncan Idaho

A meus pais, que além do amor e carinho constantes sempre incentivaram e me possibilitaram o prazer do estudo.

AGRADECIMENTOS

Agradeço a minha família, Cesare, Esther e Beti, visto que sem ela nada disto seria possível e não teria sentido.

Ao meu orientador e amigo, professor Altamiro Amadeu Suzim, pelo cuidado e atenção que dedicou a este trabalho, e pelo seu espírito pensador, que me ensinou a importância da reflexão e da inteligência quando da realização de um trabalho.

Ao Bel. Gilberto Marchioro, pela dedicação e habilidade com que implementou o editor simbólico, ferramenta importante sobre a qual se assenta parte desta dissertação.

Ao auxiliar de pesquisa e engenheirando Alexandre Hessler, pela paciência e cuidado com que realizou o levantamento de dados para as fórmulas de avaliação elétrica.

Aos professores Ricardo Reis e Dante Barone, por haverem acreditado neste trabalho e cedido valiosos recursos humanos de seus projetos para participação nesta dissertação.

Aos professores Sergio Bampi e Raul Weber, pelas valiosas críticas a este trabalho, bem como pela agilidade com que realizaram a leitura em tempo tão reduzido.

Ao professor Charles Trullemans, pela gentileza de ter aceito o convite de participar da banca da dissertação, pelas suas importantes considerações em relação ao trabalho e pela sua enorme boa vontade em suplantar as barreiras impostas pela língua.

Aos professores do CPGCC que contribuiram com minha formação, seja na sala de aula seja nas discussões informais na roda do cafezinho.

Ao pessoal da biblioteca, pela gentileza e eficiência com que sempre fui tratado.

Aos colegas do CPGCC, da turma velha (da engenharia elétrica) até aqueles que conheci somente no último ano, pela amizade e companhia em tantos almoços no RU, jantas, festas, churrascos e chimarrões juntos.

A meus amigos, que direta ou indiretamente influenciaram sobre minha vida, em especial ao Leo pela paciência e pelo chimarrão.

Finalmente, mas sem estar em último, gostaria de agradecer a Jaqueline Wendland, por tudo que fizemos juntos nestes anos.

SUMARIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

ASIC	aplication specific integrated circuit
CAD	computer aided design
DRC	design rule checker
ERC	electrical rule checker
FF	flip-flop
ggcel	gerador-gerador de células
GME	grupo de microeletrônica
GPO	gerador de partes operativas
HAD	human aided design
IC	integrated circuit
LDS	linguagem de descrição simbólica
MPC	multi project chip
PAC	projeto auxiliado por computador
PC	parte de controle
PLA	programable logic array
PO	parte operativa
RAM	random acess memory
RC	resistor - capacitor
reg_op	registrador - operador
ROM	read only memory
RT	register tranfer
TG	transmission gate
VLSI	very large scale integration
ULA	unidade de lógica e aritmética
USR	usuário

RESUMO

Este trabalho descreve uma ferramenta de implementação automática, o Gerador de Partes Operativas. A ferramenta encontra-se inserida em uma metodologia de projeto, que por sua vez é voltada para uma certa classe de circuitos.

Primeiramente, é estuda a metodologia, assim como são tecidas considerações em relação ao projeto automático de sistemas.

A busca de modelos de sistemas digitais eficientes, sua formalização e uma proposta de método de implementação são também abordados. Através de estudos em relação a diferentes implementações de algoritmos em silício surge a realização de diferentes circuitos, que serão a base da ferramenta.

Finalmente, é apresentada a ferramenta, que tem como características básicas a independência de tecnologia, a parametrização elétrica e topológica e a avaliação elétrica embutida. Os procedimentos que lograram atingir estas características são detalhados, apresentando-se exemplos de utilização da ferramenta.

ABSTRACT

This work describes an automatic implementation tool, the Gerador de Partes Operativas (data path generator). The tool belongs to a design methodology, which is tuned to a certain class of citcuit.

The methodology used is studied, and some considerations over the implementation problem are presented.

The search for efficient digital systems models is also studied, and a proposition for their automatic implementation is formalized. Different implementations of algorithms in silicon lead to different circuits, whose study is the base for this tool.

Finally, the tool itself is showed, having technology independence, electrical and compositional parameters and an embbebed electrical evaluator. The steps used to reach these features are shown, as well as examples of the use of the tool.

1 METODOLOGIA E CIRCUITOS INTEGRADOS

As possibilidades para implementação de circuitos digitais aumentaram exponencialmente com a evolução da tecnologia de fabricação de circuitos integrados. Já se projeta não apenas mais um circuito, mas sim um sistema completo para processamento de informações. A porta aberta pela elevada escala de integração de circuitos (VLSI) permite que projetistas de sistemas alcançem novas arquiteturas e máquinas cada vez mais eficientes. O preço a pagar por esta nova fronteira é o aumento da complexidade de um projeto, o que implica em tempos e custos de desenvolvimento também proporcionalmente maiores.

Para enfrentar a crescente complexidade de circuitos VLSI é necessária uma metodologia formal e estruturada, de modo a se organizar as diferentes técnicas e passos de projeto em um ambiente de Projeto Auxiliado por Computador (PAC). Mais ainda, o caráter formal da metodologia contribui com aumento de produtividade, melhoria de documentação, aumento da compreensão sobre o problema e, principalmente, automatização de todo ou da maioria dos passos do processo de concepção.

1.1 Complexidade e VLSI

É comum referenciarem-se circuitos VLSI como sistemas complexos, apontando-se isto como a causa do enorme tempo de projeto. É importante porém definir-se complexidade, de modo a se ter uma avaliação qualitativa de circuitos VLSI.

Complexidade em VLSI refere-se tanto ao número de elementos quanto as suas relações. O objetivo é classificar-se circuitos de acordo com seu grau de complexidade. Carlo Séquin (/SEQ 83/) fornece dois termos, a "complexidade explícita" e a "complexidade implícita". A primeira sugere que dois sistemas são comparáveis em complexidade quando, descritos em uma mesma linguagem,

exigem a mesma medida de custo (medido em tempo, número de páginas,etc) para sua compreensão ou especificação. Assim, a complexidade explícita de um processador é maior do que a de uma memória.

A complexidade implícita avalia sistemas que não podem ser descritos com a mesma linguagem. Por exemplo, um somador digital parece mais simples que um analógico, visto que grandezas digitais são mais facilmente manipuláveis do que grandezas analógicas.

No âmbito dos circuitos integrados, é mais interessante colocar-se uma nova visão da complexidade, dividindo-a em dois termos ou medidas, a complexidade de ordem e a complexidade algorítmica (/CAR 88a/, /SUZ 88/). A primeira considera apenas o número de elementos do sistema, e não a relação entre eles. Assim, um circuito com 5K transistores é mais complexo do que um de 3K transistores; um circuito com 77 estados é mais complexo do que um de 20, etc.

Um exemplo típico de circuitos com complexidade de ordem são as atuais memórias de 1 ou 4 Mbit, processadores sistólicos e processadores de arrays. Apesar da maior parte do circuito ser composta pela repetição de uma célula básica, o desenvolvimento desta é tarefa complexa, pelos problemas de área, consumo, atraso de linha, capacidade de corrente e outros que aparecem quando se trabalha com um grande conjunto de elementos.

A complexidade algorítmica, por outro lado, é medida pela máquina de estados finitos equivalente, incluindo alfabeto de entrada, saída, número de estados e função próximo estado. A máquina de controle de um processador é portanto algoritmicamente complexa. Um contador binário de 20 bits que poderia ter um número equivalente de estados e um alfabeto de entrada similar (uso de carga paralela) é muito mais simples. Basta observar-se que a função de próximo estado é trivial,

podendo ser descrita pelo trecho abaixo:

```
Qi(t+1) = if LOAD then INPUT
          else Qi(t) + 1;
```

Chega-se ao ponto em que tanto um processador quanto uma memória são complexos: o primeiro com complexidade algorítmica, a segunda com complexidade de ordem. Uma metodologia para concepção de CIs VLSI deve ser capaz de atacar tanto problemas com complexidade de ordem quanto algorítmica, de modo a prover recursos para acelerar ou mesmo viabilizar o processo de concepção. As técnicas aplicadas a cada um dos casos são, entretanto, diferenciadas. Enquanto a complexidade de ordem pode ser atacada pela ação hierárquica das ferramentas de layout, a complexidade algorítmica exige mais do sistema de CAD, como será comentado adiante.

1.2 A Metodologia

Método é uma palavra derivada do grego méthodos, que significa caminho para se chegar a um fim. Metodologia em concepção de circuitos integrados, no caso, é o estudo dos diferentes caminhos para se chegar a um circuito físico que opere conforme a especificação de engenharia inicial. Um caminho pressupõe um ponto inicial e final, e alguém deve percorrer o caminho de um ponto até o outro. Este alguém é o usuário do sistema de PAC, e a seqüência de passos por ele executada para percorrer o caminho é o processo de concepção. A estrutura por onde se desenvolve o processo chama-se níveis de descrição.

1.2.1 Níveis de Descrição

O caminho contínuo do processo de criação é dividido em um série de passos discretos ou níveis. Um nível é um conjunto coerente de primitivas que podem descrever um objeto, no caso, o circuito integrado. A definição da linguagem que caracteriza o nível é a própria definição do nível.

Cada nível descreve uma determinada visão do circuito integrado, sendo esta tão mais abstrata quanto mais alto for o nível. Quanto mais as primitivas de uma dada linguagem forem associadas a fenômenos físicos, mais baixo será o nível. Portanto, os níveis de transistores e de máscaras estão entre os mais baixos , visto que ambos possuem, respectivamente, atributos elétricos e geométricos.

A medida que se sobe na estrutura de níveis, tem-se visões cada vez mais abstratas do circuito, na medida em que as particularidades físicas vão sendo desconsideradas. Por exemplo, o nível lógico está acima do elétrico, pois enquanto neste trabalha-se com tensões analógicas (contínuas), no primeiro podem-se utilizar dois valores apenas, que são mapeados em 1 e 0. Necessariamente paga-se um preço pela abstração. Algumas regras devem ser estabelecidas para que quando se trabalhe no nível elétrico ao invés do lógico garanta-se o funcionamento do circuito. Para uma implementação com circuitos do tipo TTL, por exemplo, entre outras regras a seguir não se deve ultrapassar o fan-out de 10 portas por saída.

A figura 1.1 mostra 8 níveis de descrição da metodologia de concepção sendo utilizada. Cada nível é caracterizado por uma linguagem, e busca-se sempre que estas sejam formais e não ambíguas. A não ambiguidade garante que, para qualquer pessoa ou ferramenta que estude uma descrição de um circuito em um certo nível, a visão do objeto seja única. Uma operação qualquer pode ter várias implementações (algorítmica, sistólica, pipeline, puramente combinacional, etc), mas deve ser sempre encarada conforme sua descrição no nível em que foi definida. Por exemplo, pode-se definir "multiplicação=A+B", e agora o termo multiplicação significa uma soma, perdendo o siginificado usual que tem em lingua portuguesa.

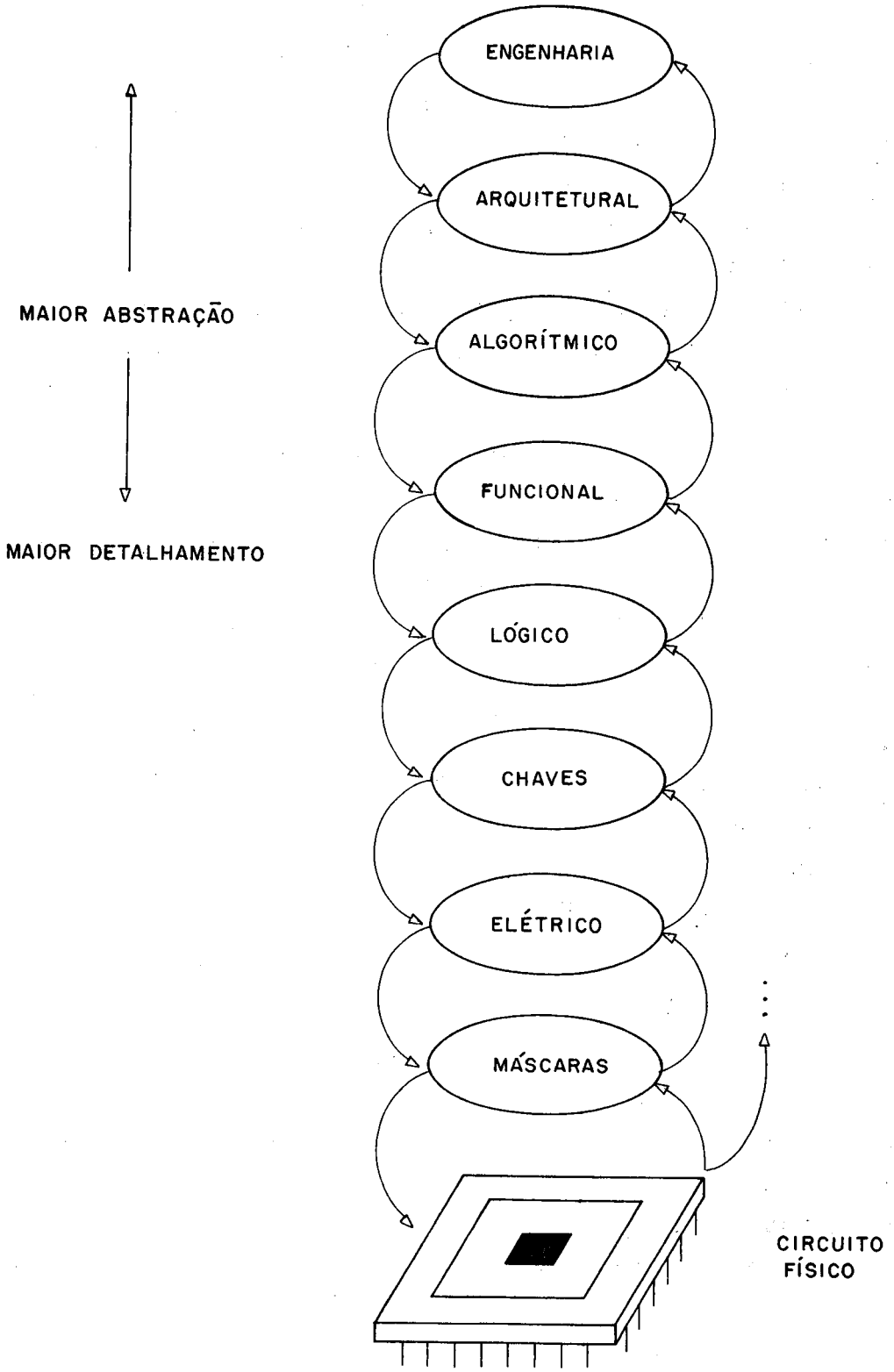


Fig. 1.1 - Níveis de Descrição

O formalismo interessa também para que seja possível a implementação de ferramentas que colaborem no processo de concepção automática, visto que o computador é

uma máquina que gera, a partir de uma linguagem de entrada, uma outra linguagem de saída.

Contraditoriamente, o primeiro nível da estrutura é o mais informal, no caso, o nível de especificação de engenharia. Em outras palavras, é a especificação do problema. Como o português (ou inglês, francês, qualquer lingua conhecida) é naturalmente uma lingua ambígua e não formal, o risco de se desejar um dado sistema e descrever-se outro é grande. Existem contudo termos e grandezas que ajudam a cercar o problema. Por exemplo, seja um circuito controlador de display de catodo comum, consumo máximo de 40mW e freqüência de varredura de 1MHz. A especificação do consumo é suficientemente formal. A da freqüência exige o conhecimento do termo varredura de maneira unívoca, assim como deve-se ter como de uso comum a nível de engenharia as palavras catodo e display. Como não existem linguagens formais para especificações de projetos de CIs neste nível, resta o uso normal do Português acrescido dos devidos termos técnicos de conhecimento comum, ou de um formulário onde são preenchidos os requisitos como em uma grande tabela.

O primeiro nível realmente formal é aquele chamado nível de sistema ou arquitetural. As primitivas aqui descrevem processos, memórias, processadores e a interconexão entre eles. O tipo de linguagem utilizada deve permitir a explicitação de paralelismo, sincronismo e concorrência entre processos. Um simulador para esta linguagem poderia localizar dead-locks ou redução de desempenho nas comunicações entre processos. Para descrição de um processador é interessante o seu conjunto de instruções, forma de entrada/saída, modos de interrupção, ciclos de acesso à memória, etc.

O próximo nível é o algorítmico. Neste ponto do projeto o que se quer fazer é descrito através de um programa ou algoritmo. Linguagens que parecem adaptadas

para descrições neste nível são as linguagens de programação tradicionais como Pascal ou C. A limitação destas decorre de sua própria origem sequencial: não é permitida uma expressão clara de paralelismo. Mais ainda, devem-se incluir construções que permitam a colocação de restrições e atributos físicos desejados sobre o circuito final. Estas restrições podem ser colocadas neste nível ou simplesmente serem herdadas de níveis superiores. Um exemplo de linguagem que comporta as necessidades comentadas é encontrado em /CAM 85/, onde se descreve a linguagem DSL.

Pode-se desejar por exemplo um algoritmo de classificação que não tome mais de um segundo para ordenar uma lista de 1000 elementos, ou uma memória que não dissipe mais de 15 mW ocupando área de 800x1000 micra quadrados. Estas especificações além de documentarem as necessidades de projeto já neste nível indicam para as ferramentas automáticas (ou até para os próprios projetistas humanos) o que se deve buscar ou mesmo o estilo de projeto a ser adotado. Assim, se for necessário implementar uma multiplicação de dois números inteiros de 16 bits em menos de 50 ns em tecnologia CMOS canal 2 micra, claro está que um algoritmo do tipo desloca-soma não poderá ser utilizado.

A expressão "if A=(B*C) then ..." no nível algorítmico pode ser interpretada como um fio ou sinal (se já houve um cálculo anterior da expressão) ou como todo um algoritmo. No nível seguinte (nível funcional) o detalhamento esclarecerá esta questão. As primitivas deste nível são estruturas e comportamentos próprios do hardware, ou seja, trabalhar-se-á com elementos típicos de circuitos, tais como registradores, ULAs, contadores, etc. Claro está que além dos elementos deve-se incluir na descrição a conectividade entre eles e a seqüência de operações impostas.

Este nível é chamado de funcional tendo em vista a relação de transformação de dados entre as entradas e as

saídas. Neste nível existem componentes (tipo de elemento e conexões) e operações sobre estes componentes (o que é feito e quando, escrita e leitura, programação da ULA, etc). As primitivas deste nível permitem descrever tempo no sentido de sincronismo entre operações. É aqui que aparece explicitamente o conceito de relógio, com todas suas possíveis fases.

Em relação às primitivas de freqüência, área e consumo, elas continuam a existir, sendo avaliadas neste passo ou transferidas a níveis inferiores. Seguindo-se o exemplo do multiplicador de 16 bits, escolhendo-se uma implementação combinacional para satisfazer o tempo, em níveis inferiores será feita a verificação se não houve desobediência em relação às exigências de área e consumo.

Novas restrições físicas são agregadas neste nível. A decisão de um PLA estático ou dinâmico tem repercussões na área, freqüência e consumo de um circuito, assim como na temporização, no número de fases a ser utilizado, na capacitância de entrada e em outros fenômenos físicos associados à construção propriamente dita do circuito.

Existe uma relativa concordância com respeito aos níveis seguintes (lógico, chaves, elétrico e máscaras), visto que desde os primeiros projetos de CIs eles foram utilizados. Na indústria de projetos semi-custom com "gate" "array" e "standard cells" as linguagems de entrada de ferramentas de posicionamento e roteamento são em sua quase totalidade baseadas no nível lógico.

O último nível é o do circuito físico final, e para este não existe uma linguagem de descrição, e sim uma pastilha de silício. Poder-se-ia extrapolar o conceito de nível encarando o circuito como uma linguagem de silício. Sobre esta pastilha serão realizados testes para validação e verificação de todo o processo de síntese. Observe-se que o circuito depois de pronto pode atender a uma série de

especificações e a outras não. Por exemplo, o algoritmo planejado é executado com sucesso somente abaixo de uma certa freqüência.

É importante frisar que os níveis aqui descritos não são fixos nem estanques. Alguns projetos podem não passar por certo nível; há casos em que outros níveis são incluídos para permitir (ou facilitar) o processo de síntese ou captura. São citados aqui termos como nível elétrico, nível lógico, etc. Para efeito de implementação, entretanto, os níveis terão nomes próprios e serão definidos por uma linguagem formal. Sem isto seria impossível evitar ambiguidades de interpretação de uma decrição. De mais a mais, citar um nível e não oferecer uma linguagem para que os circuitos sejam descritos naquele nível é totalmente inócuo. Ao invés de nível elétrico, dir-se-á nível SPICE ou ARAMOS; nível RS, EMHIR ou CIF para máscaras, etc.

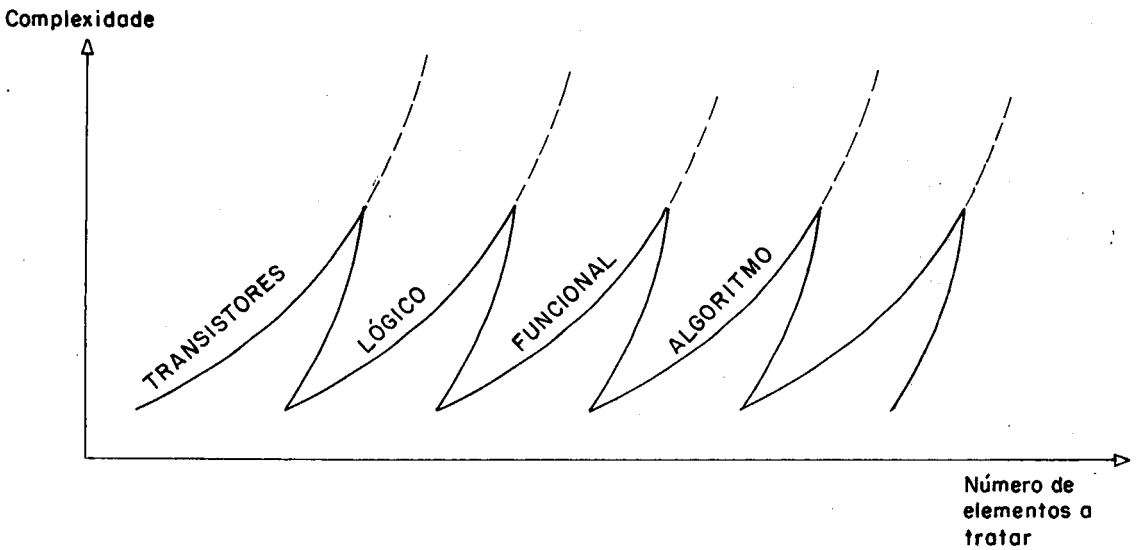


Fig. 1.2 - Complexidade e Níveis

Os níveis de descrição cumprem a função principal de permitir uma estrutura hierárquica onde se pode aplicar a regra de dividir para vencer. Os níveis são a chave para

combater a complexidade dos circuitos VLSI. E o que mostra o gráfico intuitivo (figura 1.2) adaptado de outro feito para complexidade em desenvolvimento de programas /WAT 87/.

Pelo gráfico pode-se verificar que o agrupamento das primitivas de um nível para formar primitivas de outro diminui o número de elementos de um projeto. E claro que a manipulação de um menor número de fatores facilita a compreensão e solução de problemas. Por exemplo, trabalhando-se no nível elétrico, um pequeno aumento da complexidade corresponderia a um grande aumento no número de elementos que trata o nível. Quando se faz a abstração de que 8 transistores formam um registrador CMOS, naturalmente diminui-se o número de elementos, passando-se a trabalhar em outro nível.

1.2.2 O Processo de Concepção

A possibilidade de decomposição das fases de um projeto através dos níveis de descrição permite a formação de um substrato por onde se pode desenvolver o processo de concepção. Este nada mais é que um caminhamento fundamentalmente descendente pela estrutura de níveis de abstração através de refinamentos sucessivos até chegar-se ao circuito físico. Este pode ser testado para comprovação dos resultados do projeto.

Em um processo de implementação através de refinamentos sucessivos, a cada passo (nível) interessa o quê se faz, e não o como. No nível algorítmico discute-se o andamento de um programa, e não como ele é implementado; no nível funcional quando se utiliza uma ULA não se discute em detalhes o modo de implementação da cadeia de carry; no nível lógico a maneira de se implementar um conjunto de funções booleanas pode ser através de circuitos estáticos ou circuitos dinâmicos tipo dominó, etc. Portanto, o que é realmente significativo a cada nível é o que se faz no nível, e não como um nível inferior implementa uma dada

operação.

No modelo do processo de concepção adotado, a cada nível são aplicadas ferramentas de implementação, validação, especificação e captura, conforme a figura 1.3. Na maioria dos casos de projeto, as ferramentas citadas são os próprios projetistas. Pretende-se evoluir para um processo em que não só as ferramentas, mas também quem as aplica seja uma máquina.

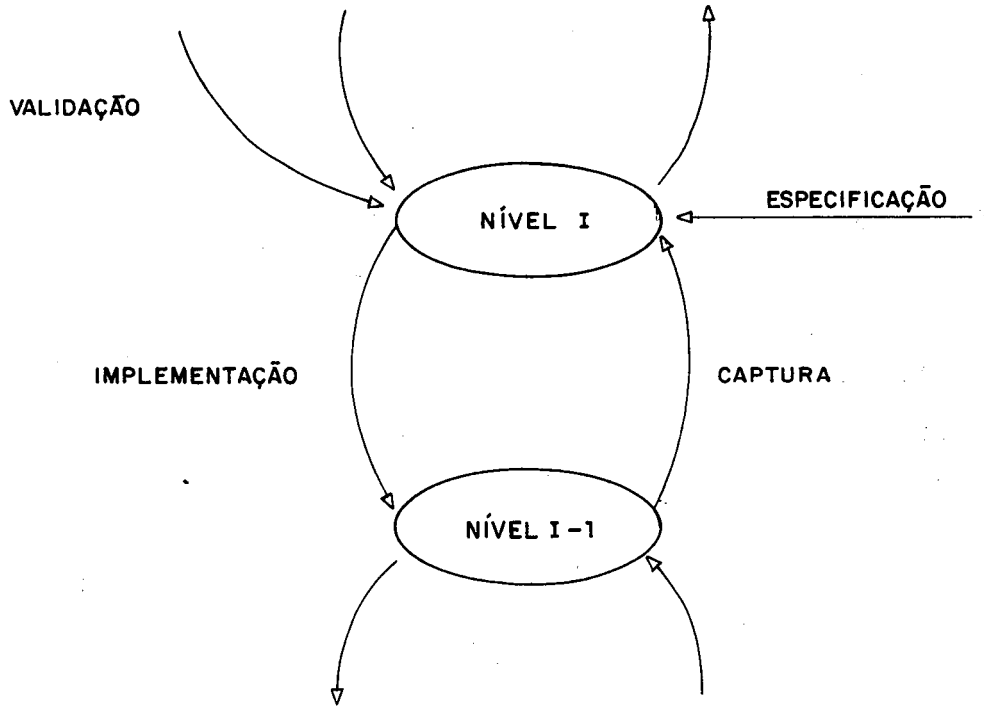


Fig. 1.3 - Componentes do Processo de Concepção

Imaginando-se a concepção como uma atividade baseada em refinamentos sucessivos, a implementação é a passagem de um nível de maior compreensão para outro de maior detalhamento. Esta passagem supõe que novas informações devem portanto ser agregadas ao nível inferior, visto que este terá mais elementos sendo um nível com mais

peculiaridades físicas. Esta inclusão de informações poderá ser feita por um projetista humano, por uma máquina ou por ambos. O ato de implementar pressupõe uma habilidade típica de projetistas, que não é facilmente transportada para uma ferramenta automática. Estas portanto devem ser abertas para que se possa realizar um ajuste fino ao projeto. Este é o conceito do HAD, ou Human Aided Design.

Gajski (/GAJ 86/) identifica ainda as seguintes tarefas dentro de um processo de implementação:

- planejamento;
- refinamento;
- otimização;
- propagação de restrições;

Durante o planejamento, imaginam-se estratégias possíveis para implementação de um determinado nível. Por exemplo, pode-se realizar um multiplicador com diferentes estilos de projeto, dependendo do algoritmo a ser utilizado, pois cada um terá um desempenho, área e consumo particular, devendo-se escolher aquele que cumpra as restrições impostas por níveis superiores.

Uma estratégia é um conjunto ordenado de passos de refinamento, otimização e avaliação. O refinamento é a arte de se atingir uma estrutura do nível inferior que atenda a necessidade de uma primitiva chamadora do nível superior. Note-se que a operação A=B+1 de um certo nível poderia ser implementada com uma ULA ou com um contador; a escolha de um ou outro é tarefa do refinamento realizado por quem efetua a descida de nível.

Uma otimização pressupõe que se possam escolher primitivas do nível inferior, para adequá-las às condições impostas. Assim, supondo-se que durante o processo de refinamento decidiu-se por um contador, resta saber se este poderá ser assíncrono ou síncrono, com "carry-look-ahead" ou não, dinâmico com duas ou quatro fases, etc.

A propagação de restrições é o ato de se

transportar para níveis inferiores informações quanto à qualidade da solução. Por exemplo, dizer que a operação A=A*2 deve acontecer em 50ns é passar a informação até o nível elétrico, onde por um processo de validação verificar-se-á se efetivamente o circuito escolhido corresponde ao desempenho esperado.

Não somente o processo de propagação de restrições é útil no que tange a avaliação de desempenho. Muitas vezes deve-se tomar decisões arquiteturais sabendo-se das possibilidades de sua implementação ou não. Utilizando-se o modelo de concepção proposto, isto caracteriza-se por uma consulta ao nível inferior.

Planejando-se a arquitetura de um microprocessador decidiu-se, por exemplo, que somente seriam implementadas diretamente em hardware as instruções LOAD, STORE, AND e NOT. As instruções mais complexas como soma e ou-exclusivo seriam interpretadas por uma ROM interna. Sabendo-se a quantidade de instruções e sua codificação, pode-se facilmente calcular o tamanho da ROM a ser utilizada. Uma ROM enorme inviabilizaria esta proposição para a arquitetura do microprocessador.

É preciso que, quando se utilizam ferramentas de implementação automática, o problema de uma avaliação a priori de problemas de níveis inferiores seja levado em conta. Por exemplo, o gerador de ROMs do GME (/CAR 88b/) quando chamado pode devolver o tamanho e o atraso da ROM sem gerar as máscaras. Este esquema de consultas permite não só um caminhamento pela estrutura de níveis sem se ter de abandonar o atual, mas também uma maneira organizada de se avaliar projetos durante o seu desenvolvimento. No futuro (espera-se próximo), quando programas realmente eficientes estiverem substituindo os projetistas humanos nos níveis superiores, aqueles também utilizarão o esquema de consultas de modo a avaliar as possíveis estratégias de projeto e estilos disponíveis.

Quando se faz uma descida de nível que não de maneira automática, erros podem ser inseridos na descrição do nível inferior. É necessário portanto o ato de validação, ou seja, uma comprovação de que a descrição está correta. Para garantir esta correção, pode-se proceder de três maneiras distintas.

A primeira delas seria pela comparação da descrição com outra correta por hipótese. Este é o caso por exemplo de um circuito implementado com standard cells, onde se pode fazer uma comparação do net-list das células utilizadas com aquelas descritas no nível lógico, garantindo que a passagem deste nível para o de máscaras foi bem feita.

Pode-se também validar um sistema pela verificação do uso correto das primitivas de um certo nível, utilizando-se programas como DRCs (/GOM 88/), ERCs (/MEE 88/), verificadores sintáticos ou semânticos. Por exemplo, no nível funcional deve-se assegurar que não se escreve em uma ROM, ou que duas ULAs não colocem seus dados no barramento ao mesmo tempo; no nível elétrico não deve existir uma porta sem alimentação; no nível de máscaras não se ferem as regras de projeto, etc. Os métodos de verificação mais utilizados hoje em dia restringem-se a níveis inferiores (os já citados DRCs, comparadores de net-list, ERCs, etc).

No caso de uma ferramenta de implementação automática, assumindo-se que a passagem de um nível a outro é correta por construção, então não são necessárias ferramentas de validação. Contudo, sempre que existir um projeto em que houver intervenção humana, mesmo que somente para um ajuste fino (HAD), a possibilidade de inclusão de erros na descrição faz-se presente.

A prova formal de correção está ainda em fase exploratória, e não é atualmente, pelas informações disponíveis, ferramenta pronta para uso. Desta maneira, a

maior parte do trabalho de validação é feito geralmente com o uso de simuladores, ou seja, compara-se o conjunto de saídas obtidas com o conjunto de respostas esperado para um determinado padrão de entrada. Não se testa apenas uma primitiva, mas o seu conjunto.

A eficiência da simulação está diretamente relacionada com uma correta escolha do padrão de testes apropriado ao simulador e ao circuito. Prova-se a correção apenas para o conjunto de excitações fornecido. No caso de ser impossível a simulação para um conjunto de entrada exaustivo, existirão dúvidas quanto ao comportamento da descrição naquele nível para o conjunto de entrada. Detectam-se apenas os erros esperados, e não aqueles escondidos, geralmente mais críticos. Geradores automáticos de vetores de teste podem facilitar o trabalho; seu campo de aplicação atual restringe-se na sua grande maioria aos níveis lógico e elétrico.

A captura pode ser útil para um processo de validação: se a descrição em um dado nível consegue resolver o problema proposto, então um retorno (automático ou não) ao nível superior deve produzir como resultado um conjunto de primitivas equivalente àquele que deu origem ao nível de partida. O problema será então verificar a equivalência da descrição.

Assim, uma vez estando pronto o circuito, se com uma câmera de TV fosse feita a captura das máscaras através das geometrias visíveis no "chip", poder-se-ia verificar, antes mesmo de alimentar o circuito, se houve erro de fabricação (falta de um contato, conexão aberta ou em curto, etc), pelo simples confronto do resultado da captura com a descrição supostamente correta do objeto.

Quanto mais se sobe na hierarquia, mais complexas se tornam as ferramentas de captura. Do nível funcional ao algorítmico pode-se chegar a várias descrições com mesmo comportamento, pelo simples fato de se utilizar diferentes

primitivas da linguagem (FOR ou WHILE podem ser comandos iterativos equivalentes).

Uma vez que se possua em mãos o CI implementado, pode-se proceder a fase de testes físicos. O teste não serve apenas para verificar se o circuito funciona ou não, mas deve ser encarado como o último passo para validação final do processo de concepção. Uma etapa de fabricação isolada (metalização por exemplo) é de difícil teste a nível de circuito pós-encapsulamento. O teste permitirá validar indiretamente esta etapa.

É através de ensaios sobre o circuito físico que se comprovam especificações impostas, desde a freqüência mínima de operação até o correto sequenciamento de estados do autômato finito. O contrário também é possível: algumas vezes o circuito possui o comportamento esperado, mas não se atinge a frequência máxima desejada, ou consome-se muita potência, enfim, não se atingiram as características físicas esperadas. Pode-se acrescentar que os projetistas de ferramentas de PAC utilizarão o circuito inclusive para validar as ferramentas e o processo de concepção. Caso não haja conformidade entre o previsto e o medido dentro de um nível de tolerância aceitável, procurar-se-á melhorar o modelo.

O circuito depois de construído também é um nível de descrição. Fazendo-se o caminho inverso (por captura) pode-se chegar até a compreensão do comportamento de um determinado circuito. Utilizando-se um circuito conhecido, é possível a validação de todos os níveis percorridos pelo processo de concepção, podendo-se aplicar a ele padrões de teste do nível lógico, elétrico, funcional, etc. O próprio circuito assume o papel anteriormente desempenhado pelo simulador. Assim, aplicando-se as excitações do nível elétrico (sinais), a saída deve corresponder à esperada; aplicando-se um conjunto de instruções, a saída deve estar no conjunto de códigos esperados, etc. Obviamente, os padrões utilizados para verificação nos diversos níveis

deverão ser transpostos a um nível em que sejam compatíveis com o nível de circuito, ou seja, elétrico.

Uma vez que os modelos de circuito utilizados a cada nível de abstração estejam corretos, o teste não serve mais para comprovação do projeto em suas fases de implementação, mas sim para verificação de falhas de fabricação. Testa-se um circuito no nível elétrico por ser mais econômico. Poder-se-ia examinar as pastilhas fabricadas uma a uma com uma câmera de TV, mas isto seria muito demorado. Uma inspeção total do circuito a cada fase de fabricação também seria muito onerosa. No limite, quando os modelos encontrarem-se com um grau de tolerância aceitável, o teste valida apenas uma passagem de nível: a manufatura.

2 CONCEPÇÃO AUTOMATICA

2.1 O Porquê do Projeto Automático

A microeletrônica poderia ter seu espectro de atuação ampliado, visto que a tecnologia evolui rapidamente, e mais e mais transistores podem ser integrados. Além disto, a competitividade do mercado de eletrônica exige que os projetos sejam cada vez mais eficientes, rápidos e seguros, com a devida personalização que dificulta a cópia indevida de sistemas. O mercado de sistemas eletrônicos confia cada vez mais nos ASICs, ou Aplication Specific Integrated Circuits, onde os próprios engenheiros de sistema tem acesso à implementação de circuitos diretamente em silício.

As afirmativas acima enfrentam o problema de que, para projeto de sistemas mais complexos, o tempo de desenvolvimento será grande, sendo necessária a presença de especialistas de projeto em silício. A relação entre o número de engenheiros de sistema para o número de engenheiros de silício é muito grande, provocando, além da falta de tempo para se passar um sistema para silício, um problema de disponibilidade de mão-de-obra (/DEM 87/).

Para permitir o avanço da microeletrônica sobre as áreas de projeto de sistemas é preciso fornecer aos projetistas de sistemas meios de acesso à microeletrônica. Estes meios não são recursos humanos altamente especializados em microeletrônica, mas sim são ferramentas para construção de circuitos em silício. Estas ferramentas tem o papel de não somente servir ao engenheiro de sistema como também devem ajudar a diminuir o tempo de projeto. Dependendo da eficiência com que as ferramentas implementam circuitos, os próprios projetistas de silício utiliza-las-ão, ou seja, elas serão tão eficientes que atingirão o grau de exigência dos projetistas de silício.

Para atender a esta necessidade, o mercado de ASICs (Application Specific Integrated Circuits) possui

ferramentas bastante difundidas para projeto de circuitos "semi-custom" em "gate-array" ou "standard-cells". Com baixo "turn-around" e baixo número mínimo de circuitos a fabricar, estes sistemas são confiáveis e largamente utilizados. Vários fabricantes possuem pacotes de CAD (/VLS 86/) que exigem do engenheiro de sistemas apenas o conhecimento do próprio sistema e do uso de ferramentas relativamente simples como simuladores e roteadores. O uso destas ferramentas permite que se tenha acesso ao potencial da microeletrônica sem conhecê-la a fundo, bastando conhecer como funcionam os simuladores lógicos e elétricos para avaliação funcional e de desempenho.

Se o uso de "gate-arrays" e "standard-cells" resolve o problema da difusão da microeletrônica, não resolve aquele provocado pelo aumento da complexidade devido à possibilidade de integração de dezenas de milhares de transistores. Isto porque é inpensável projetar e simular a nível lógico um circuito com 10.000 portas, devido não só ao tempo de projeto dispendido como à dificuldade de se manter o projeto validado e correto quando se trabalha com um número grande de primitivas. Pesquisadores japoneses estimaram em mais de 100 homens/mês o custo de um projeto lógico de um circuito "standard-cell" que levou apenas 2 homens/mês para ter o layout finalizado (/SAN 85/).

Soluções semi-custom possuem um uso não ótimo de área, tendo-se também (ou até em decorrência) um baixo desempenho em termos de velocidade, consumo e potência. Em /DEM 86/ encontra-se uma ilustração entre as relações de complexidade e performance de circuitos, repetida na figura 2.1.

Para posicionar-se na região mais central do gráfico (região A), novas técnicas e ferramentas de CAD são necessárias, que são os geradores de módulos e os compiladores de silício. No momento em que se conseguir

atingir esta região, pode-se oferecer ao projetista de sistemas ferramentas eficientes que realmente coloquem todo um sistema complexo em uma ou mais pastilhas, sem perda de performance e com baixo tempo de desenvolvimento.

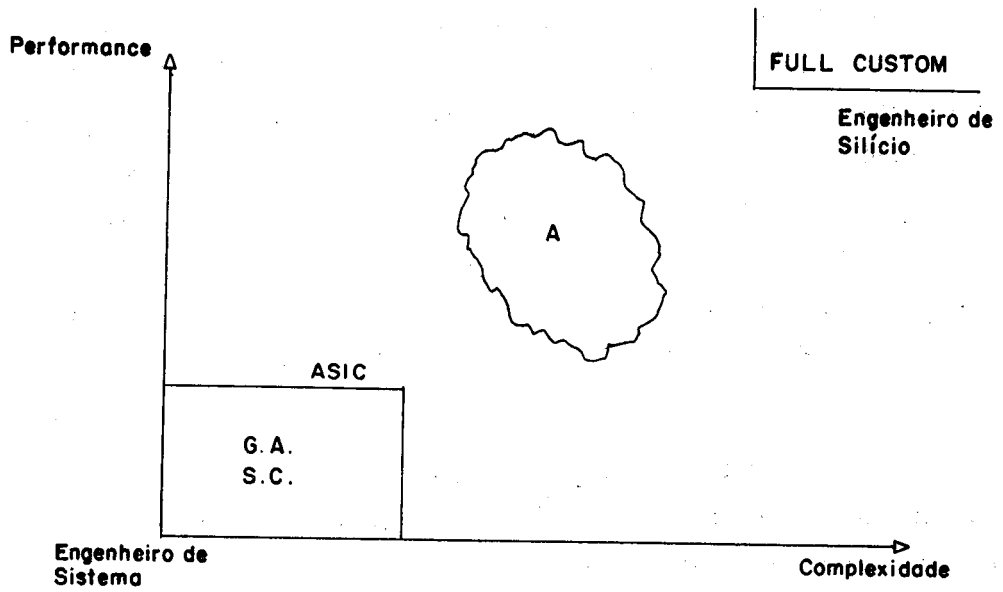


Fig. 2.1 - Complexidade e Performance

2.2 O Compilador de Silício.

Metodologias com "gate-arrays" ou "standard-cells" partem de componentes básicos para montar uma estrutura, como quando se agrupam portas para formar um flip-flop, ou células para fazer um contador. O projeto é tipicamente "bottom-up", e depois de pronto deve-se trabalhar a nível de portas lógicas para avaliação elétrica, lógica e uso do roteador.

Este tipo de abordagem ao problema da concepção de circuitos integrados tem a desvantagem de não explorar a natureza hierárquica do projeto nas suas ferramentas, possuir funções fixas e pouca liberdade elétrica e topológica, gerando um resultado final desotimizado.

Os compiladores de silício tentam suplantar estas desvantagens justamente explorando a natureza hierárquica do projeto. O termo Compilador de Silício foi introduzido

em 1979, designando sistemas que montavam o layout de circuitos a partir de células pré-desenhadas (Bristle Blocks, /JOH 79/). Hoje os compiladores são encarados como tradutores de um nível de descrição mais abstrato para layout, conforme /GAJ 86/. Em uma classificação presente em /PAN 87/, os compiladores, dependendo de suas linguagens de entrada, podem ser estruturais ou comportamentais. Os primeiros recebem como especificação de entrada um conjunto de componentes e sua interconexão. No segundo caso, o projeto inicia pelo fornecimento ao compilador de um conjunto de portas de entrada e saída, o comportamento destas umas em relação as outras e um conjunto de restrições.

Conforme vinham sendo pensados, os compiladores de silício seriam a solução para projetos de alta complexidade, baixo volume, baixo tempo de desenvolvimento e alto desempenho, como na região A da figura 2.1. Contudo, tanto a síntese quanto o projeto físico são tarefas complexas, envolvendo restrições, compromissos e possibilidades muito grandes e entrelaçadas para serem integradas em uma única ferramenta.

Analisando-se publicações sobre os compiladores de silício, verifica-se que aqueles que se preocupam com a parte de síntese (/PAN 87/, /KAP 86/, /JAM 86/, /SIN 87/, /SMI 86/, /RAB 85/, /HIT 83/, /SHR 82/, /JHO 85/ entre outros) tecem poucas considerações quanto as diversas possibilidades de layout, assumindo que este será realizado através de uma biblioteca de células e programas de posicionamento e roteamento automático.

O inverso também acontece: compiladores de layout não fornecem modos de entrada mais alto do que uma descrição estrutural do circuito a ser gerado, geralmente amarrando a topologia do sistema a uma arquitetura pré-definida (como em /LIN 87/, /ROW 87/, /RUE 86/ e outros). São efetivamente poucos os sistemas que procuram

atacar, ou ao menos possibilitar a manipulação de compromissos nos diversos níveis de projeto. Reporte-se os compiladores de arquitetura fixa (/ROS 85/ e /DEM 86/), assim como o sistema da empresa SDL em /BUR 86/, em que a responsabilidade da construção do compilador (ou pelo menos a passagem de algoritmo para o nível funcional) fica a cargo do usuário, sendo fornecido um pacote de geradores de módulos e simuladores.

De qualquer maneira, existem comercialmente ou a nível universitário desde geradores de células (/LIN 87/, /RAP 87/, /NOG 85/, /FRI 85/, /KIM 84/), de módulos (/ROW 87/, /BUR 86/, /BAR 85/, /HUN 85/, /WIL 84/), processadores (/SIS 82/, /JER 86/) e sistemas (/DEM 86/). O problema está em se verificar quão próximos das reais necessidades dos usuários finais estão estes programas. Ou seja, serão eles realmente satisfatórios?

Em /EVA 85/ comenta-se que apesar de muito falados, os compiladores de silício são pouco comercializados. Isto possivelmente se deve a dois motivos: como são ferramentas novas, não existe uma tradição dos engenheiros de hardware de tratar o próprio hardware em níveis mais abstratos; são ferramentas complexas e fechadas, não oferecendo muita versatilidade aos projetistas de hardware, que não conseguem exprimir com facilidade seu problema (não estando acostumados ao uso de formalismos) nem influenciar significativamente na solução.

Baker, da Prime Computer Inc. (/BAK 86/), relata a experiência de sua equipe de projetistas de sistemas que adquiriram um compilador de silício. Seus objetivos eram o baixo tempo de desenvolvimento do produto final (um novo computador), baixos custos de projeto e fabricação, pequena área ocupada, pouco consumo e alta performance. Além disto, não se desejava manter uma equipe de especialistas em layout, mas sim conter o conhecimento adquirido no âmbito da própria firma. Após análise, foi escolhido um compilador que oferecia um conjunto de geradores de funções (PLA, RAM,

ROM, Data Path), um avaliador de arquiteturas (para que, sem chegar a detalhes do projeto, se tivesse uma idéia do desempenho) e uma interface com o usuário eficiente e poderosa, de modo que o compilador não precisasse ser utilizado por especialistas de circuitos e layout.

Pela experiência comentada observa-se que o usuário, ao mesmo tempo em que deseja automatizar a transferência de um projeto baseado em circuitos de prateleira para silício, também deseja ter um certo controle sobre a ferramenta que utiliza. Este controle significa a possibilidade de escolher a melhor arquitetura, o circuito mais veloz, a menor área, enfim, movimentar o projeto no plano de concepção da figura 2.2. É pela falta de ferramentas que forneçam tantos graus de liberdade ao usuário que os compiladores de silício ainda não se tornaram o carro chefe da indústria de ASICs. Além disto, o compilador deve estar integrado em um ambiente de CAD já estabelecido e com uma metodologia de projeto à suportá-lo.

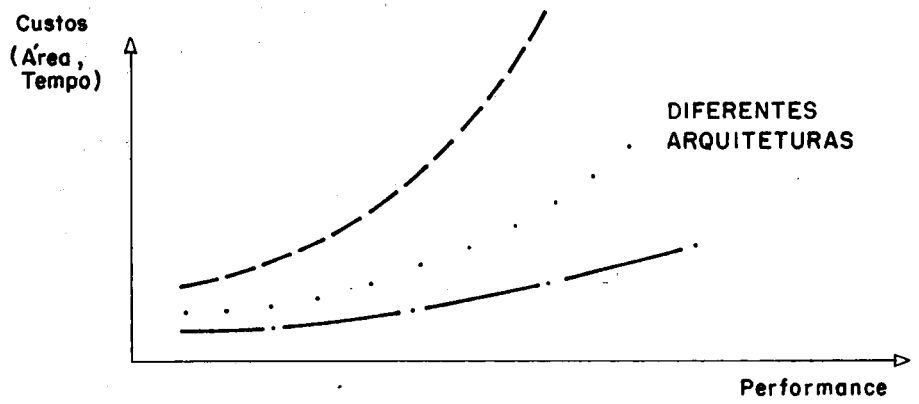


Fig. 2.2 - O Plano de Concepção

2.3 Considerações sobre um Compilador Aceitável

Um compilador de silício ideal deveria ser capaz de automatizar todo o proceso de concepção, produzindo um circuito que cumprisse com os compromissos de área, velocidade e consumo impostos pelo usuário. A linguagem de

entrada para este compilador seria abstrata o suficiente para que um ciruito integrado complexo pudesse ser expresso em poucas linhas ou páginas nesta linguagem. Um compilador assim ainda não existe, e o caminho que leva a tal programa é longo, pela enorme quantidade de conhecimento necessária para implementação de um circuito complexo. Envolve-se desde a arquitetura do circuito a regras de projeto, da alocação de registradores à velocidade da ULA, inúmeras variáveis entram em jogo tornando o compilador ideal um sonho distante.

Uma vez que se defende uma estrutura hierárquica de concepção, então o compilador de silício deve também tomar parte desta estrutura, substituindo o projetista no que tange o processo de concepção, caminhando sobre os níveis de descrição anteriormente expostos.

A tarefa de passagem de um nível para outro inferior é trabalho altamente especializado e baseado em heurísticas particulares. Para certa classe de problemas, algumas soluções padrão são eficientes, outras não. Um mesmo problema, com diferentes restrições de área e frequência pode exigir maneiras totalmente distintas de se passar do nível algorítmico para o funcional, ou do funcional para o lógico, ou deste para as máscaras, etc. É preciso ter em mente que a divisão em níveis de descrição hierárquicos tem a função de organização do processo de concepção, mas uma descida de nível quase sempre envolve conhecimentos não apenas dos dois níveis envolvidos, mas principalmente da função de mapeamento de um nível para outro, além das implicações existentes (ou provocadas) em outros níveis.

Por exemplo, dado um algoritmo, como implementá-lo em silício de maneira automática? Deve-se decidir o grau de paralelismo do hardware, as máquinas de estados do controle, o tipo e quantidade de operadores, a quantidade de registradores em função do número de variáveis, o tipo de interconexão, a estratégia de

sincronização, a freqüência máxima, etc.

Para implemetar uma ULA, deve-se ter em mente a quantidade de área disponível, que também é função do tipo e quantidade de operações desejada, da carga que deve ser suportada na saída, da estratégia de sincronização utilizada, etc. Para se trabalhar com toda a gama de variáveis, os engenheiros de CAD que trabalham em compilação de silício sempre fizeram alguma restrição sobre alguma fase do projeto. Alguns definem uma arquitetura alvo fixa, como os sistemas MacPitts e Syco (/SIS 82/, /JER 86/), outros perfazem a síntese sem levar em conta todas as possibilidades do silício, como anteriormente mencionado, assumindo uma biblioteca de células e um programa de posicionamento e roteamento (/VLS 86/).

Em um artigo recente, Pangrle e Gajski (/PAN 87/) propõem um compilador de silício inteligente, centrando esforços na tradução de comportamento para estrutura com a utilização de componentes microarquiteturais diferentes ou estratégias de roteamento diferentes. A proposta parece correta, tendo em vista a enorme disponibilidade de biblioteca de células e de programas de posicionamento/roteamento no momento.

Este estilo de solução tende a otimizar a parte arquitetural de um circuito, mas não a passagem para o silício. No momento em que se utilizam bibliotecas de células mais poderosas em termos de disponibilidade de circuitos e estratégias de roteamento, tem-se um espaço de projeto em silício maior, mas não será garantida uma ocupação ótima de área e freqüência, sem contar as dificuldades presentes quando se tem de manipular uma biblioteca que, para ser útil, deve ter várias células. Para tal, seriam necessárias ferramentas de implementação capazes de explorar a variedade de recursos e soluções que um projeto VLSI oferece.

Contrariamente à construção de um circuito,

propõe-se (dentro da metodologia aqui utilizada) que a construção de ferramentas de CAD obedeça a um processo "bottom-up", onde se capturam as dificuldades de níveis inferiores para depois seguir aos superiores com modelos e resultados eficientes. Desta maneira, cada ferramenta domina o seu nível, podendo fornecer a níveis superiores respostas precisas quando chamadas.

Kuuttila (/KUU 85/) afirma, embora sem quantificar, que a maior quantidade de esforço e tempo em termos de calendário para um projeto full custom é gasto em layout. Isto é bastante compreensível, visto que para o projeto de uma simples célula é necessário desenhá-la, executar o programa de DRC e o extrator, simulá-la e redesenhá-la até que tenha as características desejadas.

A discussão acima, aliada ao ao fato de que seres humanos possuem habilidades para aplicar soluções baseadas em heurísticas (como alocação de registradores, recursos de conexão e operadores), permite imaginar que a abordagem para se atacar o problema da complexidade em VLSI passa primeiro pela implementação de ferramentas que atuem em níveis mais baixos de projeto. Quando estes possuirem um conjunto de ferramentas adequado, e quando houver conhecimento suficiente para se transportar a heurística de projeto de circuitos para a máquina, então integrar-se-á o sistema como um compilador de silício realmente eficiente. Para tal, imagina-se que toda ferramenta construída deve ser suficientemente aberta para receber indicativos da melhor solução (HAD), e deve ser também aberta para que possa ser modificada e ampliada para uma nova classe de problemas, de maneira que a cada novo projeto em que é utilizada se possa aprimorá-la.

A grande energia do projeto deve ser gasta nos níveis superiores da hierarquia, onde a criatividade e a habilidade humana são mais necessárias. É importante liberar o projetista de tarefas que exijam alta dose de

esforço braçal, mas sem perda significativa de desempenho global, que abrange área em silício, velocidade e potência.

Este trabalho propõe uma ferramenta que deve colaborar para construção de um compilador de silício ideal. Características como linguagem de entrada de um nível de alta abstração, fácil movimentação no plano de concepção, HAD e liberação do usuário de tarefas cansativas foram buscados desde os primeiros instantes de desenvolvimento. Procurou-se fornecer ao usuário características de uma ferramenta ideal, voltada também para desempenho, para que quando de sua ligação a um sistema integrado de concepção o PAC fosse o mais próximo possível daquele desejado pelo usuário.

3 SISTEMAS DIGITAIS EXPRESSOS ALGORITMICAMENTE OU ALGORITMOS IMPLEMENTADOS ATRAVES DE SISTEMAS DIGITAIS

Um sistema digital pode ser visto como um transformador do conjunto de sinais binários de entrada X em um outro conjunto binário Y. A transformação, quando feita com o uso de circuitos puramente combinacionais, possui suas saídas em qualquer instante de tempo dependentes apenas das entradas naquele mesmo tempo (figura 3.1). A transformação ou operação Y=op(X) é função apenas dos atrasos existentes nos componentes físicos de op(). O problema desta aproximação será o alto custo de recursos que um sistema maior tende a exigir.

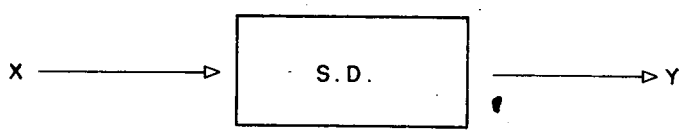


Fig. 3.1 - Sistema Digital

Como nem sempre é necessária esta ordem de velocidade de resposta, pode-se utilizar o conceito de sequencialização, de modo a compartilhar-se no tempo recursos dispendiosos em termos de área, consumo, etc. Aplica-se então o conceito de sistemas digitais sequenciais (figura 3.2), onde para o conjunto de entradas {X,E} de um dado instante, produz-se o conjunto {Y,E+1}, onde E significa o estado atual do sistema. O estado é quem memoriza a informação necessária para a seqüência das operações envolvidas nas transformações de dados para as quais o sistema foi projetado.

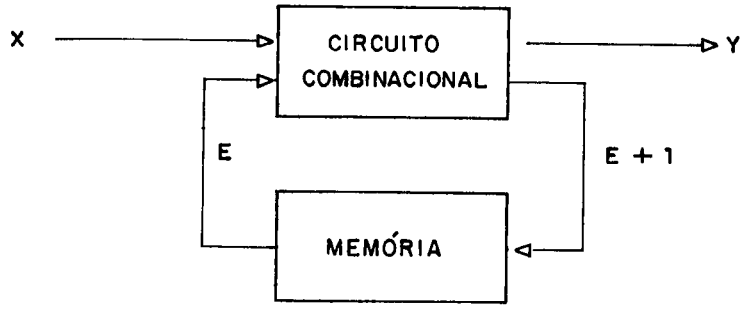


Fig. 3.2 - Sistema Digital Sequencial

Escreve-se $(Y_1, E_1) = op_1(X_1, E_0)$; $(Y_2, E_2) = op_2(X_2, E_1)$; e sucessivamente até $(Y_m, E_m) = op_m(X_m, E_{m-1})$, onde X_i é o conjunto de dados de entrada do sistema no instante i, Y_i é o conjunto de dados de saída para o mesmo instante, E_i é o estado i sendo E_{i-1} seu estado predecessor, e op_i é o conjunto de operações que deve ser executado sobre os dados no estado atual. O que interessa ao sistema é a passagem de um estado à outro até o final da computação.

Conforme exemplo encontrado em /YAS 86/, seja a função $y:\{0,1\}^n \rightarrow \{0,1\}^n$, $y=\Pi^n_{i=1} Xi$, $Xi \in \{0,1\}$, $1<=i<=n$. Duas possíveis implementações funcionalmente equivalentes (por produzirem o mesmo resultado) serão analisadas. A primeira, puramente combinacional, encontra-se na figura 3.3. O tempo de resposta é função direta da tecnologia para implementação, devido ao atraso de cada porta utilizada.

A alternativa para esta implementação encontra-se na figura 3.4, onde o registrador R memoriza o resultado de uma computação anterior, enquanto os sinais Xi entram serialmente no sistema digital. Agora, a resposta da computação depende não somente de parâmetros tecnológicos, mas também da sincronização do sistema. Anteriormente, o número de elementos de X influenciava no atraso máximo do circuito combinacional. Com o circuito da figura 3.4, a

influência de X é no número de estados necessários para a computação de y.

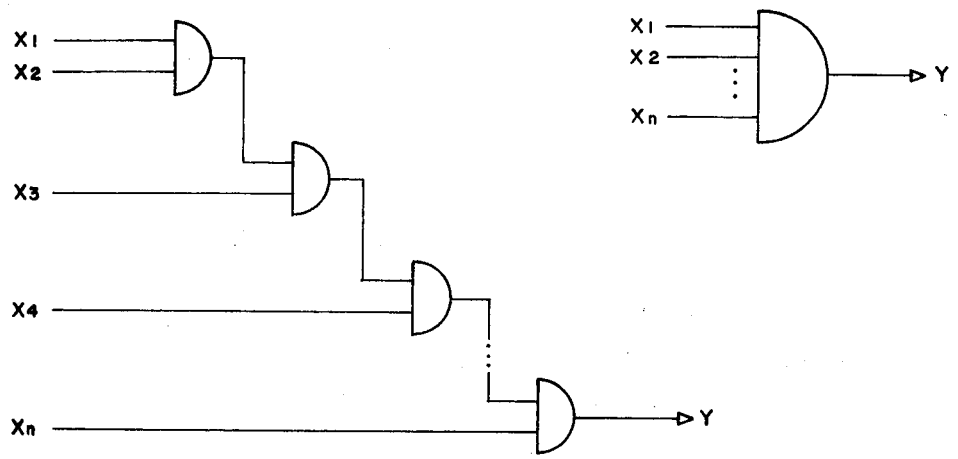


Fig. 3.3 - Implementação Combinacional

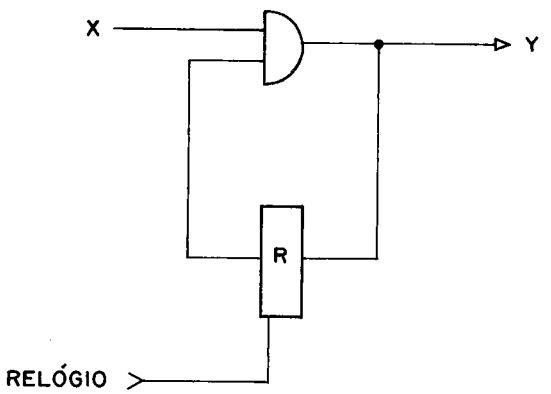


Fig. 3.4 - Implementação Sequencial

Se para a transformação passa-se de um estado E0 para um estado En, qualquer que seja a ordem intermediária de estados, com ou sem laços, existe intrinsicamente a noção de programa, como uma sequência ordenada de operações. Sendo assim, um sistema digital sob forma de máquina sequencial pode ser descrito como um algoritmo.

A conclusão é óbvia: se sistemas digitais sequenciais podem ser descritos por um algoritmo, então existem linguagens em que, escrevendo-se o algoritmo desejado, especifica-se por extensão um sistema digital associado /SUZ 81/. A derradeira expressão do algoritmo é o sistema digital que o implementa.

No exemplo fornecido anteriormente, a construção

```
R:= True;
For i:=1 To n R:=R AND Xi;
```

expressa de maneira algorítmica o circuito da figura 3.4. Pode-se então, a partir de uma descrição de um certo algoritmo, implementar o circuito digital equivalente. Sabe-se que é uma tarefa factível e comumente executada por projetistas humanos. A dificuldade encontra-se na automatização deste processo de concepção. A realização sem intervenção humana de um circuito só é possível no momento em que modelos adequados de circuitos encontram-se disponíveis para serem trabalhados.

Modelos corretos não são suficientes de per si na tarefa de concepção de circuitos integrados VLSI. Não se pode iniciar o projeto de um circuito de 10 mil transistores pelo nível de máscaras, assim como é impensável um cronograma que suporte o tempo de desenvolvimento de um processador de 100 mil transistores que deva ser totalmente projetado a partir do nível lógico. Ambos os modelos para transistor e agrupamento destes podem estar corretos, mas são totalmente inadequados para a tarefa em questão.

Entende-se que o modelo adequado de sistema digital é aquele que, sendo formal, permite uma visão clara e facilmente manipulável para se chegar à implementação física, seja ela feita pelo homem ou por uma máquina. Um modelo é tanto melhor quanto maior o grau de eficiência, compreensibilidade e factibilidade tiver o circuito que ele permita alcançar. O modelo da figura 3.2 pressupõe que um

sistema digital seja uma máquina sequencial, e como tal é um avanço em relação ao modelo da figura 3.1, visto que sistemas sequenciais síncronos ou assíncronos permitem realizar circuitos fora do alcance daqueles combinacionais, como por exemplo, um simples contador.

Para circuitos de maior porte, é conveniente utilizar-se um modelo um pouco mais complexo, que é o da divisão de sistemas digitais em Parte Operativa e Parte de Controle. Segundo Davio (/DAV 83/), já em 1965 Glushkov concebeu o comportamento de um sistema de cálculo pela cooperação de dois subsistemas, o operacional e o de comando.

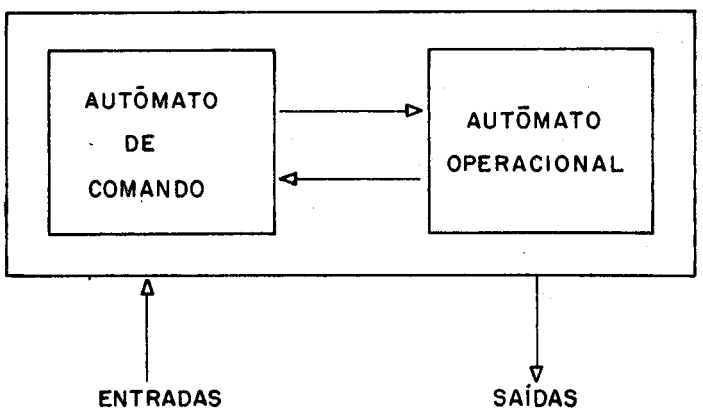


Fig. 3.5 - O Modelo de Glushkov

Visto do exterior, o sistema da figura 3.5 transforma um alfabeto de entrada E em um alfabeto de saída S. Esta transformação é efetuada com o uso de um conjunto de operadores. A parte operativa pode ser considerada como um autômato onde o conjunto de estados é o conjunto de configurações possíveis dos caracteres de entrada no decorrer do algoritmo. Os operadores, atuando sobre estes caracteres, transformam o estado da parte operativa.

A parte de comando pode ser representada por um autômato onde as saídas correspondem a ordens de ativação

de operadores e memórias da parte operativa e, eventualmente, das saídas externas.

O funcionamento do sistema pode ser esquematizado da seguinte maneira: a parte operativa recebe da parte de comando algumas ordens que determinam a execução de certas ações, que em resposta devolve à parte de comandos informações sobre o estado das operações em curso ou realizadas. Estas informações tem o nome de variáveis de condição.

O importante desta caracterização é que tanto a parte operativa quanto a parte de controle são encarados como máquinas de estados finitos, e como tal podem ser descritos através de um algoritmo como já foi mencionado. Como um algoritmo é um sistema digital, este pode por sua vez ser decomposto em PC-PO, tendo-se então uma recursividade de aplicação do conceito de parte de controle-parte operativa para uma dada descrição algorítmica.

Concretamente, este conceito implica que um algoritmo pode ser decomposto em PC-PO, cada uma destas partes sendo por sua vez passíveis de serem decompostas em PC-PO novamente, como mostra a figura 3.6. Este aninhamento é útil para a economia de recursos físicos e para estruturação do projeto. Um algoritmo que calcula o máximo divisor comum de dois números pode ter a parte de divisão feita através de um circuito combinacional ou de um sub-autômato com estados suficientes para tal.

A constatação da recursividade na decomposição PC-PO é importante porque possibilita, a partir de uma descrição inicial de um algoritmo, o processo de implementação automática do sistema digital. Este será chamado de Interpretação (/SUZ 81/), baseado na seguinte proposta: a expressão de um algoritmo A em uma linguagem L é representada A/L, sendo isto um programa P na linguagem L. O mesmo algoritmo pode ser implementado fisicamente

sobre um dado meio material. A realização física de A é representada por A/F, e é uma máquina M como na figura 3.7. O objetivo é a construção de máquinas algorítmicas, e para tal, não se deve considerar cada circuito como um caso a parte, mas sim como diferentes realizações sobre uma mesma estrutura de base. Esta é uma analogia perfeita da codificação de diferentes programas escritos com uma mesma linguagem de programação. Não existe linguagem ótima para todas as aplicações; os circuitos assim implementados serão adaptados a uma dada classe de problemas.

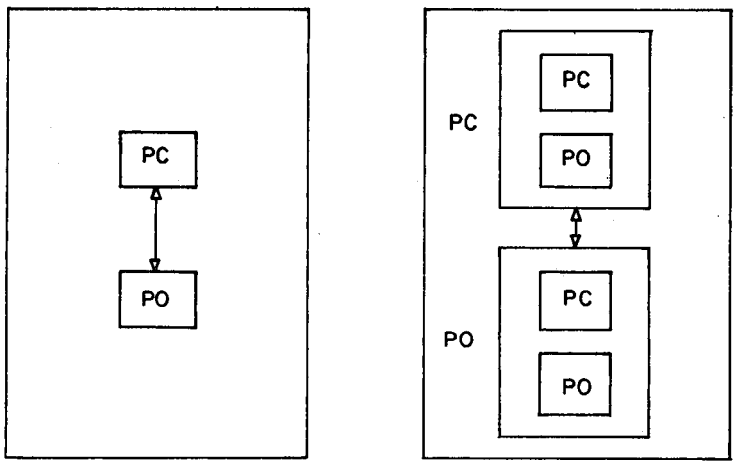


Fig. 3.6 - Decomposição Recursiva

Um interpretador I_i de uma linguagem L_i de um certo nível i é um algoritmo capaz de executar todo programa escrito nesta linguagem. O interpretador em si pode ter sido escrito em uma linguagem L_{i-1}, repetindo-se o processo de interpretação até o último nível desejado (figura 3.8). O nível final possui o interpretador I_0, que é implementado fisicamente, representando uma máquina M capaz de executar, indiretamente, as instruções de L_n através de interpretações sucessivas.

Observe-se que um programa escrito em L_n deve ser interpretado de maneira que sejam aceitos tanto o programa quanto as variáveis e os dados que ele manipula. Tudo isto é o conjunto de dados do interpretador, que por sua vez

terá de utilizar-se de variáveis de trabalho internas, formando uma cadeia ao longo de todos os níveis de interpretação.

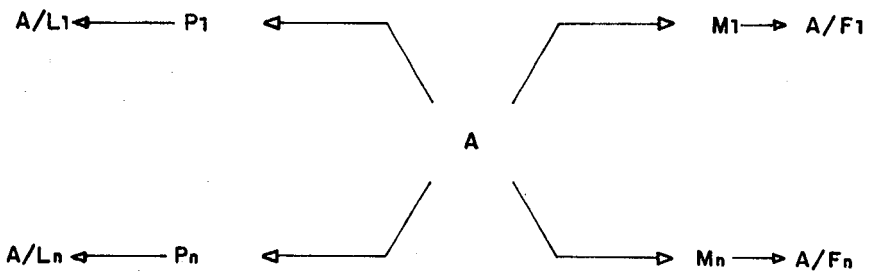


Fig. 3.7 - Algoritmos e Máquinas

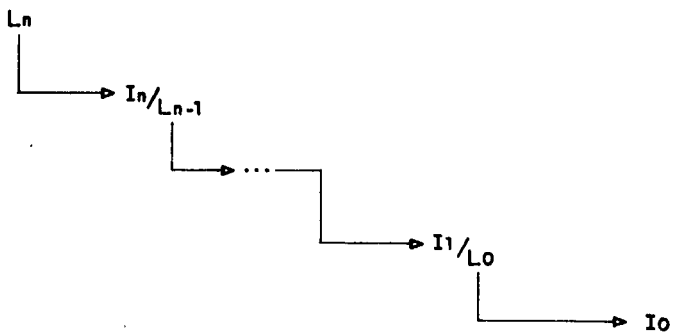


Fig. 3.8 - Processo de Interpretação

Para um dado nível i com linguagem L_i podem existir diversos interpretadores que levem a L_{i-1} com características diversas. Por exemplo, a construção

```
For i:=0 To 20
Do A:=A+1;
```

pode ser interpretada de forma a produzir como resultado a construção

```
i:=0;
L1: If i>20 Goto L2;
A:=A+i;
i:=i+1;
Goto L1;
L2: ... ,
```

que por sua vez necessitaria da interpretação do comando If por um interpretador de nível mais baixo.

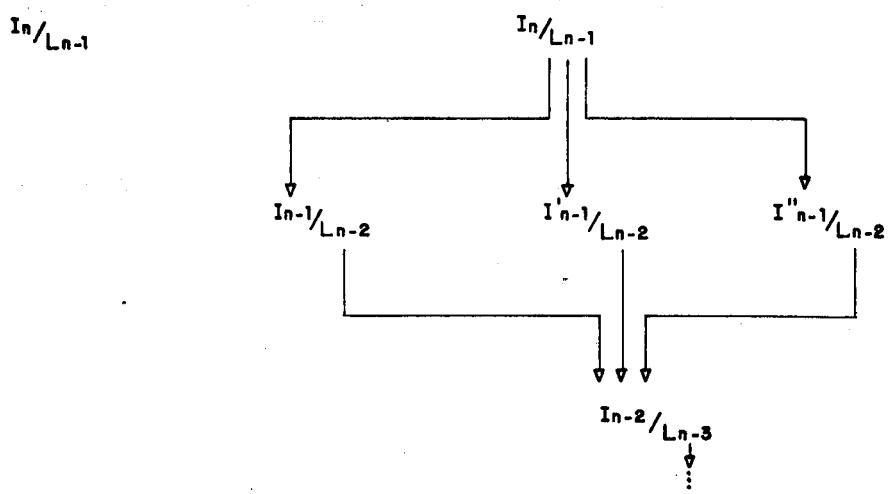


Fig. 3.9 - Interpretadores: notação e encadeamento alternativo

O processo de interpretação pode ser colocado como abaixo:

> Se a especificação é simples o suficiente,
> Então implemente a especificação em hardware,
> Senão decomponha em uma especificação mais simples e aplique o processo de interpretação sobre ela.

Tomando-se o exemplo acima, uma rápida inspeção no algoritmo indica que será necessária a constante 20 (comando i>20). Pode-se considerar que A:=A+i é uma operação suficientemente simples para ser implementada,

assim como a constante 20, mas não o If, o que implicaria em nova interpretação.

Este conceito de interpretação explica como, por exemplo, em uma descrição do nível algorítmico, a operação A:=B*C pode significar uma operação simples ou toda uma máquina de estados de níveis mais baixos, dependendo do interpretador que se escolher para passagem de nível.

Resta contudo definir o que é uma especificação simples o suficiente. Isto dependerá do interpretador que se utiliza, da quantidade de informações que este consegue gerenciar e das técnicas disponíveis para interpretação.

Um exemplo concreto de interpretação será desenvolvido a seguir. Seja o trecho de algoritmo

```
IF A!=0
THEN C:=MULTIPLICA(A,B);
...
```

Será preciso interpretar estas operações para um dado meio físico. Tomando-se como unidade de medida o número de ciclos de relógio necessários, pode-se proceder então com três possíveis casos de interpretação:

a)para uma arquitetura como a do microprocessador 68000.

A seguinte descrição de assembler do 68000 implementa o trecho de algoritmo proposto:

```
        cmpi #0, d0         /* a variável A está em d0 */
        beq  próximo        /* se zero, pula           */
        mulu d0, d3         /* C:=A*B, resultado em d3 */
próximo nop                 /* e o programa segue      */
```

Uma rápida análise do número de ciclos necessários à implementação do algoritmo em questão revela que serão necessárias 4 instruções em assembler para implementar duas de uma certa linguagem de um nível superior. Contudo, é preciso levar em conta que as instruções de assembler deverão ainda ser por sua vez interpretadas pelo conjunto 68000-memória, provocando que em hardware sejam necessários 47 ciclos de máquina mais 6

ciclos de acesso à memória, perfazendo um total de 53 ciclos de máquina, ou seja, as 4 instruções do assembler (4 estados no nível assembler) geraram 53 estados em um nível inferior capaz de interpretar o assembler. Para este exemplo, o interpretador I0 é o próprio 68000.

b)Para uma arquitetura dedicada.

Pode-se desenvolver um circuito que implemente a função multiplica pela interpretação do trecho abaixo, onde / e * significam deslocamento à direita e à esquerda respectivamente:

```
COBEGIN
i:=0;
C:=0;
COEND;
COBEGIN
  WHILE(i<16) {
    if (B[0]!=0)
        COBEGIN
                i:=i+1;
                A:=A*2;
                B:=B/2;
                C:=A+C;
        COEND
    if (B[0]==0)
        COBEGIN
                i:=i+1;
                A:=A*2;
                B:=B/2;
         COEND
  }
COEND;
```

Supondo-se um hardware capaz de interpretar estas primitivas (deslocamento à esquerda e à direita, soma e múltiplos testes em um estado), nota-se que o número de estados necessários para este nível é de somente 2 (para a função multiplica), e a multiplicação redundaria no nível inferior em 17 ciclos de máquina. Claro está que o custo

desta interpretação é superior ao anterior, visto que se deverá possuir este circuito dedicado para implementação da multiplicação. O aumento de custo reflete-se no aumento de desempenho.

c)Para uma segunda implementação dedicada.

Pode-se desenvolver um circuito que implemente diretamente a função multiplica de uma maneira combinacional. O número de estados utilizado para tal implementação/interpretação seria de apenas 1. Contudo, o custo de um circuito multiplicador combinacional e rápido é bastante elevado.

E importante porém construir-se uma família de interpretadores que leve em conta as especificações de desempenho necessárias ao projeto, tais como a área máxima permitida, a velocidade de obtenção de respostas para um dado conjunto de entradas (número de estados necessários), a freqüência máxima alcançável, etc.

3.1 Classificação de Circuitos Digitais

Embora circuitos possam ser expressos por algoritmos e decompostos em parte operativa e parte de controle, nem todos os circuitos são eficientemente projetados utilizando-se esta abordagem, visto que o modelo de decomposição PC-PO pode não mais ser eficiente. Suzim, em /SUZ 88/, apresenta uma classificação de circuitos digitais bastante útil para se decidir quando utilizar esta aproximação.

Fundamentalmente, no âmbito deste trabalho, os circuitos digitais podem ser divididos em circuitos combinacionais, contadores, memórias e circuitos algorítmicos. Os primeiros implementam gates, tabelas verdade e funções simples, podendo ser implementados em lógica anárquica ou em estruturas como ROMs, PLAs, ULAs, multiplicadores paralelos, etc. A eficiência destes circuitos depende diretamente da tecnologia utilizada, da

área ocupada e da dissipação máxima. Como não possuem elementos de memória, não existem estados internos.

Os contadores executam um número bastante limitado de funções sobre uma estrutura de dados fixa: são os contadores up, up-down, com reset, carga paralela, etc. Estes circuitos, possuem circuito de controle reduzido, sendo projetados para servirem de blocos de funções para construção de circuitos maiores.

A maior influência tecnológica se faz presente em circuitos de memória, onde a velocidade máxima e o consumo e área mínimos nunca são suficientes para a demanda do mercado. O projeto de memórias envolve desde engenheiros de silício até pessoal altamente especializado no processo tecnológico. Como os contadores, a parte de controle é extremamente reduzida, sendo quase impossível dissociá-la do resto do circuito, principalmente por ser assíncrona.

Os circuitos algorítmicos são aqueles que realmente podem usufruir do processo de concepção por interpretação. Claramente a divisão em parte operativa e parte de controle auxilia na implementação deste tipo de circuitos, onde se enquadram desde processadores comerciais de alto desempenho até controladores de vídeo, processadores RISC, processadores de ponto flutuante, aceleradores em hardware, etc.

São com os circuitos do tipo algoritmo que se montarão (montam) os grandes sistemas em silício onde, por exemplo, um microprocessador é apenas um de seus blocos. Estes sistemas serão o lugar comum em termos de concepção, e não a exceção.

É para a classe de circuitos algorítmicos que se pretende colaborar na implementação automática, através de um gerador de partes operativas base, isto é, partes operativas que não podem ser mais decompostas em PC-PO. A divisão em PC-PO é útil porque permite a estratégia de dividir para vencer: ataca-se um problema algoritmicamente

complexo por partes. Além disto, é possível obter uma maior eficiência na implementação de cada uma das partes pela aplicação de estratégias específicas que levem em conta suas peculiaridades.

3.2 Um Exemplo de Decomposição

Para ilustrar a decomposição recursiva de um algoritmo em parte de controle e parte operativa, desenvolveu-se o exemplo de um controlador de elevador, que apesar de sua aparente simplicidade possui embutidos conceitos úteis e aplicáveis a problemas maiores.

O controlador tem a seguinte especificação de engenharia:

-quando da ligação de energia, o elevador deve dirigir-se ao térreo e lá ficar, até ser chamado;

-quando houver chamada o elevador deve:

a) ir ao andar;

b) abrir a porta;

c) fechar a porta e esperar um comando;

-a qualquer instante, se apertado o botão de emergência, o elevador deve parar no próximo andar e abrir a porta.

A esta descrição informal de engenharia pode-se, ou melhor, devem-se agregar características físicas, dizendo qual a velocidade a que o circuito deve responder, qual a tensão de alimentação, qual deve ser a imunidade a ruídos, etc.

Observe-se que certos detalhes podem parecer implícitos para um observador sendo totalmente ignorados por outro, dentro deste mesmo nível de descrição. Por exemplo, o elevador ficará parado depois de atendidos todos os pedidos, repousando no último andar visitado. Este comando não foi explicitado, mas pode ser inferido pela descrição informal do comportamento do elevador.

É importante ressaltar o fato da ambiguidade ou indefinição na descrição, uma vez que podem ocorrer situações semelhantes para outros componentes do controlador. A falta de explicitação de todas as características pode levar a controladores que perfazem funções distintas. Apesar de para o exemplo este problema não parecer catastrófico, haverá situações em que ele o será. Portanto, uma especificação mais formal do comportamento do circuito faz-se necessária. Não interessa como ele será implementado, mas é fundamental que obedeça às especificações de seu comportamento esperado.

Para evitar ambiguidades será utilizada uma linguagem algorítmica fortemente baseada em C /KER 78/. A esta linguagem foi agregado o comando COBEGIN/COEND, que indica que as instruções devem ser executadas em paralelo. A partir da descrição informal anterior tentar-se-á montar uma descrição formal do controlador com esta linguagem. Procurar-se-á identificar e utilizar ao máximo possível processos paralelos para controle do elevador. A grosso modo, pode-se dizer que existem os seguintes procedimentos envolvidos no controle do elevador:

-inicialização;

-avaliar e memorizar pedidos internos;

-avaliar e memorizar pedidos externos;

-controlar o carro para subir, descer, abrir porta,etc.

Os processos físicos envolvidos são paralelos: uma memória deve registrar pedidos de usuários externos e internos, enquanto que algum processo deve ser encarregado de desligar o pedido no instante em que este for atendido. Esta é uma configuração similar àquela do problema produtor-consumidor, que aparece com freqüência em sistemas concorrentes /PET 85/. Os mesmos cuidados aplicados a sistemas operacionais devem ser aqui utilizados, tais como proteção de variáveis, sinalização entre processos dependentes, etc. É evidente que a linguagem de descrição

de hardware para este nível de descrição deverá suportar este tipo de primitivas. Sendo assim, prossegue-se com a descrição formal do controlador, disposta na figura 3.10.

```
SISTEMA elevador;
main()
{
        EXPORTACAO Reset,emergência;
        IMPORTACAO ACK_emeregência[ACK0,ACK1];
        ENTRADA Topo,Térreo,emer_BOT;
/* procedimentos de inicialização   */
        while(!Térreo)
                Reset=1;
        Reset=0;
/* procedimentos para operação normal, dois processos e uma PO base que
   trabalham concorrentemente                                          */
        while(TRUE)
            COBEGIN
                controla_carro();
                avalia_pedido();
                {
                 while(TRUE)
                    if(emer_BOT==1)
                        emergência=1;
                    while(ACK_emergência[*]!=1)  /* espera que
                                    ;          todos os processos parem  */
                    executa_procedimentos_emergência();
                }
            COEND;
}
```

Fig. 3.10 - Descrição do Elevador

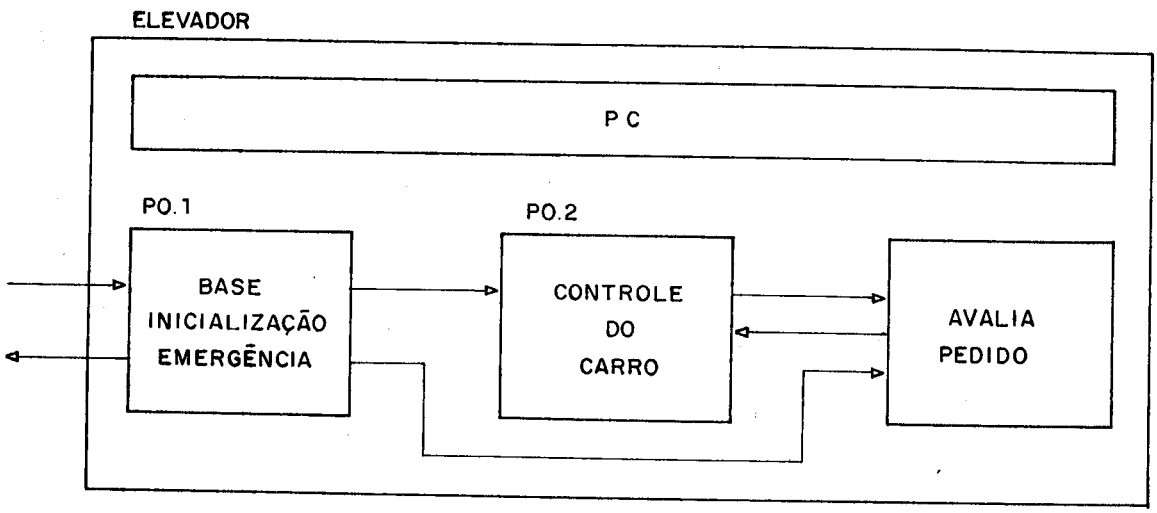


Fig. 3.11 - Circuito Elevador

Note-se que à primeira vista este é um elevador bastante simples. Após a inicialização, as funções de leitura de botões e armazenamento, controle do carro e emergência são executadas em paralelo. Imaginando-se que cada processo é um sistema digital, ter-se-iam três sistemas que poderiam utilizar uma memória comum para armazenar os pedidos, ou pode existir uma memória unicamente em um dos processos, devendo os outros sinalizá-lo para escrita-leitura na memória através de mensagens que contenham a informação necessária.

Todas estas opções refletir-se-ão no circuito final. Podem existir diversos circuitos físicos equivalentes funcionalmente. Associando-se a cada processo um sistema digital, é muito provável que o circuito final estivesse na forma da figura 3.11.

As flechas indicam as possíveis comunicações entre os processos. Estas foram colocadas a nível de subsistema. Como cada sistema digital pode ser expresso em termos de PO e PC, não está ainda claro neste ponto do processo de concepção se a comunicação será feita a nível de PC ou de PO. Esta dúvida possivelmente será sanada mais adiante, quando da descrição do interior de cada módulo. É o usuário quem vai decidir se a comunicação será através da PC (o autômato avalia a condição) ou através da PO (uma passagem de dados), expressando-se a decisão na descrição de um nível mais abaixo.

Na figura 3.12 tem-se a descrição do processo avalia_pedido(). É importante verificar que as diferentes opções de projeto já se fazem notar. A declaração EXPORTAÇÃO indica que as variáveis aqui presentes serão compartilhadas por outros módulos. Significa também que um dos meios de comunicação entre este processo e os outros será através da parte operativa. Então, possivelmente estas memórias deverão ser de duplo acesso, ou pode-se estabelecer uma pequena fila e um gerente de memória ou

processo árbitro que controle a tentativa de escrita por processos diferentes. Se realmente fosse esta a opção, a escrita e leitura seriam realizadas por processos diferentes, e portanto deve-se evitar a escrita em fila cheia ou leitura em fila vazia. No limite, pode-se imaginar uma fila composta de um registrador e um flip-flop. A escrita no registrador é feita pelo processo que deve enviar a mensagem, que também seta o flip-flop. O processo consumidor da fila lê o flip-flop, e caso este esteja setado, o registrador é lido e o FF resetado. Se o processo produtor for escrever e o FF estiver setado, então o produtor não consumiu o dado, e a escrita não se completa.

```
PROCESS avalia_pedido()
{
        EXPORTAÇÃO: mem_S[n],mem_D[n],ACK1;
        ENTRADA: botão_D[n],botão_S[n];
        ENTRADA: botão_andar[n];
        IMPORTAÇÃO: apaga_memória, end_apg_mem[n],
                    emergência, Reset;
        LOCAL i;
/* Algoritmo: fica para todo o sempre varrendo os botões internos e
   externos,verificando emergência e verificando quem deve
        ser desligado */
ACK1=0;
while(TRUE)
        {if(Reset)
                {for(i=2;i==n;i++)
                        mem_S[i]=mem_D[i]=0;    /* zera pedidos */
                 mem_D[1]=1;    /* faz pedido para descer ao térreo */
                }
         for(i=1;i==n;i++)
               {if(apaga_memória==1)
                        mem_S[end_apg_mem]=mem_D[end_apg_mem]=0;
                                /* reseta memórias */
                 if(botão_andar[i]==1)    /* se há pedido interno */
                        otimiza_direção(mem_S,mem_D,i);
                 if(botão_D[i]==1)   /* se há pedido para descer ext. */
                        otimiza_direção(mem_S,mem_D,i);
                 if(botão_S[i]==1)  /* se há pedido para subir, ext. */
                        otimiza_direção(mem_S,mem_D,i);
               }
         if(emergência==1) /* se existe emergência, para o
                 break;                         procedimento normal */
        }
ACK1=1;  /* indica para o processo emergência
                                que está parado      */
}
```

Fig. 3.12 - Processo avalia_pedido

Existe uma outra forma de comunicação entre processos a qual se pode depreender das declarações: a sinalização. O sinal apaga memória quando ativo indica que uma memória endereçada pelo sinal end_apg_mem deve ser apagada. Como não há interferência de outro processo na PO, mas apenas uma sinalização, então possivelmente este seria o tipo de comunicação utilizável por partes de controle. Se isto for assim implementado, então será preciso definir um protocolo de comunicações entre as PCs que desejam conversar. Este protocolo poderá estar distribuído nas partes de controle ou poderá estar a cargo de um terceiro autômato finito. Estes modelos de comunicação podem ser vistos nas figuras 3.13 e 3.14.

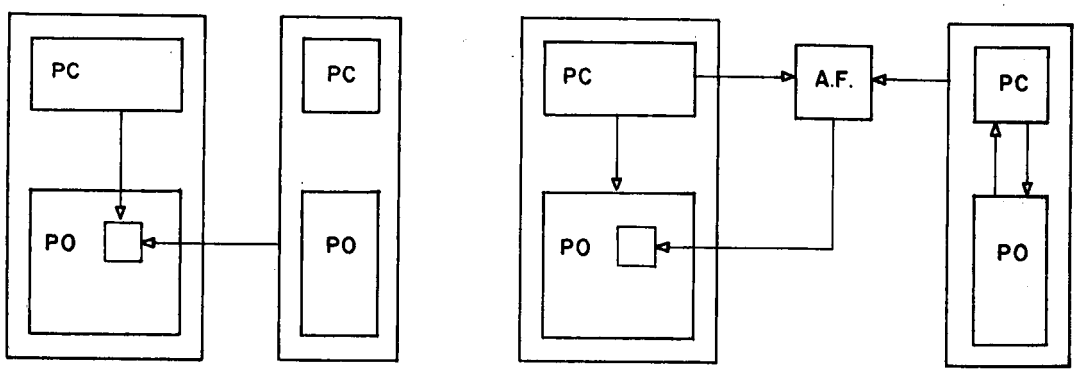


Fig. 3.13 - Comunicação através da PO

O processo controla_carro, descrito na figura 3.15 é bastante ilustrativo em termos de decomposição de sistemas digitais. Na descrição do processo definem-se variáveis de entrada, de saída, interprocessos e variáveis locais. Estas últimas encontram-se portanto na PO do sistema digital controla_carro. Mais adiante, novos processos são definidos dentro de controla_carro. Isto significa que, pela descrição, pode-se tomar a iniciativa de criar um subsistema dentro de controla_carro como por exemplo, verifica_pedido. Este subsistema também seria decomposto em PO e PC, englobando por sua vez mais um

subsistema, o processo para_andar. Na figura 3.16 tem-se um diagrama desta divisão.

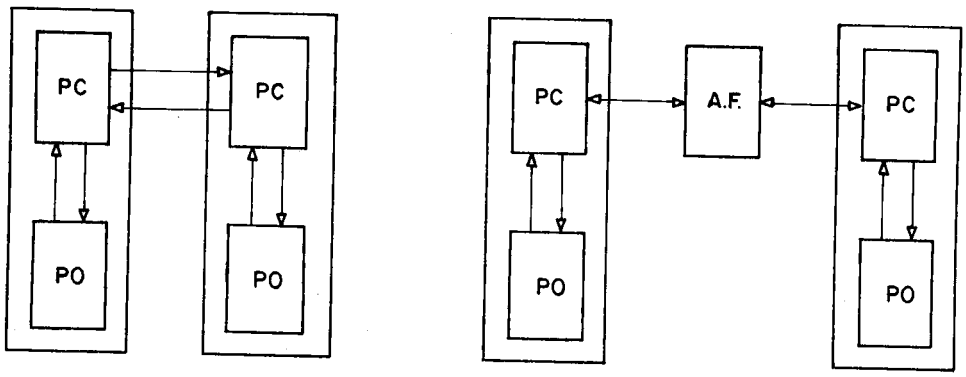


Fig. 3.14 - Comunicação através da PC

```
PROCESS controla_carro()
{
        IMPORTACAO: mem_S[n],mem_D[n], Reset, emergência;
        EXPORTACAO: apaga_memória, end_apg_mem[M],ACK0;
        LOCAL:      i, direção, andar_atual;
        SAIDA:      sobe,desce;   /* sinais para controle do motor */
        ENTRADA:    marca_andar, Térreo;
ACK0=0;
while(TRUE)
    {
       if(Reset)
              desce_terreo();
       COBEGIN
              {verifica_pedido();            /*                        */
               COBEGIN                       /* esta parte corresponde */
                     atende_pedido(direção); /* a operação normal      */
                     verifica_pedido();      /*                        */
               COEND
              }
              {if(emergência==1)
                    {
                     salva_operação_atual();
                     para_andar();
                     ACK0=1;
                    }
              }
       COEND
    }
}
```

Fig. 3.15 - Processo controla_carro

Na decomposição, todos os processos abaixo de controla_carro foram colocados em sua parte operativa, o

mesmo acontecendo com a localização de para_carro em verifica_pedido. Esta decisão explica-se na medida em que, se o processo chamador tem de chamar o processo dele dependente, esta chamada equivale a uma ordem para execução de uma operação completa, que portanto faz parte de sua parte operativa. A operação pode ser tão complexa quanto se queira, até mesmo envolvendo um subsistema digital completo para resolvê-la.

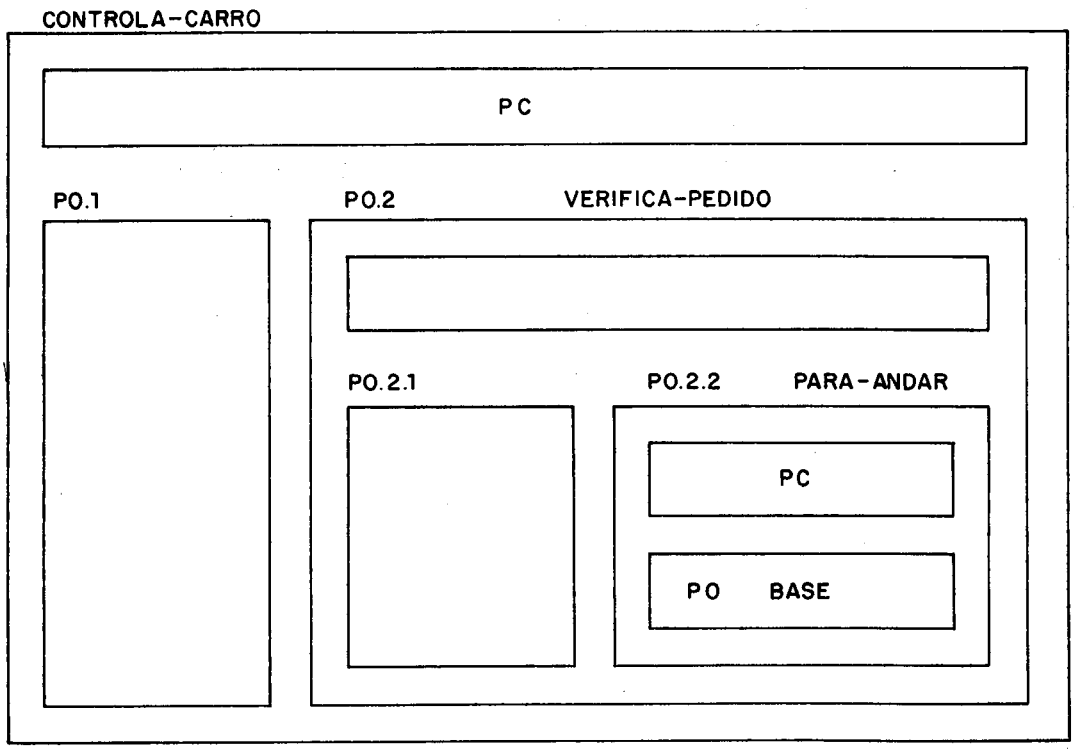


Fig. 3.16 - Divisões de controla_carro

Está claro então que a idéia de decomposição pode ser aplicada recursivamente a tantos níveis de aninhamento quanto necessário. Por exemplo, na subrotina para_andar foi definida a variável de saída porta_aberta, que será um fio indicando que a porta está aberta. Porém, esta poderia não ser uma variável a que a PO tivesse acesso, mas sim uma sinalização a outro processo para que este ativasse a porta. No caso, optou-se por uma variável indicadora, possivelmente um registrador de um bit.

Depois que todos os processos tiverem sido

descritos é possível a avaliação das partes operativas presentes. Chama-se de PO base ou fundamental aquela em que não existem mais decomposições ou chamadas de processos. Por exemplo, em para_andar existe uma parte operativa base, visto que nenhum processo é chamado; existem apenas operações sobre variáveis (porta_aberta=1, j++). Como serão feitas fisicamente estas operações é decisão do projeto a posteriori, visto a enorme variedade de opções. Um registrador-incrementador pode ser usado para construção de j++ (j=j+1), ou pode-se ter um registrador e uma ULA, um contador com carga paralela, etc. Mesmo a este ponto do projeto, cada uma das opções reflete um certo compromisso de engenharia entre custo, desempenho, facilidade de projeto e outros.

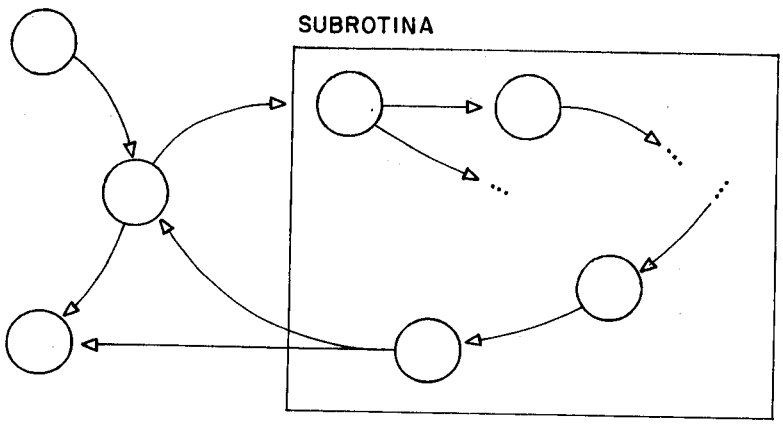


Fig. 3.17 - Autômato com subrotina

Subrotinas como desce_térreo, para_andar, etc utilizam a mesma PO da máquina de estados finitos que as invoca. A existência das subrotinas implica em um desvio da seqüência do autômato com um retorno ao ponto de partida, como na figura 3.17. Se o autômato finito fosse implementado como um microprograma, seria necessário o salvamento do PC em um registrador auxiliar. Caso sejam permitidas pela linguagem do nível chamadas recursivas ou aninhadas, o registrador passa a ser uma pilha cujo tamanho

é função da profundidade desejada de chamada de subrotinas.

Uma vez feita a descrição do sistema controlador de elevador, pode-se partir para sua implementação. Se houvesse uma ferramenta para síntese automática, esta realizaria um circuito com diversas POs decompostas em outras POs e PCs, conforme a descrição fornecida. Não é forçosamente necessário que cada decomposição consuma tempo de execução do sistema digital, isto é, torne-o mais lento. Como a PC e a PO são ambas máquinas de estados finitos, um comando da PC à PO pode ser executado em diferentes velocidades, dependendo do relógio de cada uma das máquinas. O protocolo de comunicação obedeçe as mesmas considerações feitas para comunicações entre máquinas de estados finitos.

Analisando-se manualmente a descrição, podem-se identificar partes operativas associadas a cada processo. Assim, avalia_pedido possui duas memórias de n bits, mem_S[n], mem_D[n]. Deve ser permitido acesso a cada um dos bits destas memórias, para escrita ou remoção de um 1 lógico. Como mais de um processo utiliza estas memórias, elas poderão ser de duplo acesso para leitura ou poderão estar subordinadas a um coordenador de escrita/leitura sendo acessadas por sinais, poderão ser estáticas ou dinâmicas, tendo 2^n registradores de um bit, o que implicaria no uso de um decodificador 2n:1 para endereçamento, ou poderiam ser organizadas em dois registradores de n bits.

Todas estas opções e muito mais não se encontram expressas em nenhum lugar da descrição. A chegada no circuito final é claramente um processo de implementação, em que um maior detalhamento deve ir sendo agregado ao projeto a cada instante. Mesmo para uma PO base, isto é, que não pode ser decomposta, as possibilidades de implementação são muitas. No anexo 1 encontra-se a descrição completa do Sistema Elevador.

4 O MODELO DE PARTE OPERATIVA:ARQUITETURA DE MICROCIRCUITOS

Segundo Davio (/DAV 83/), o subsistema operacional aparece como uma máquina de estados finitos (autômato operacional), denotando-se $A=\langle E,O,Q^{\alpha},M,N\rangle$, onde E é o alfabeto de entrada, O o alfabeto de saída, Q^{α} é o conjunto de estados internos, N é a função de saída e M a função de transição de estados. Para o comando c_i associa-se uma transformação $Q^{\alpha}\rightarrow Q^{\alpha}$ do conjunto de estados internos de A, e escreve-se

$$c_1:[R_1]=g1([R_{1,1}],\ldots[R_{1,\omega_1}]);$$

$$\ldots$$

$$[R_{\beta}]=gs([R_{\beta,1}],\ldots[R_{\beta,\omega_{\beta}}]).$$

Os termos à direita do sinal de atribuição referem-se ao instante de tempo imediatamente anterior ao pulso de relógio que altera o estado, enquanto que os termos a esquerda referem-se ao conteúdo dos registradores no instante imediatamente posterior a ocorrência do relógio. Um comando c_i pressupõe várias funções atuando sobre um conjunto de registradores, passando o resultado desta computação sobre um registrador destino. Este cálculo de funções associado ao fato de que um registrador à direita pode também aparecer à esquerda do sinal de atribuição exige a presença de uma barreira temporal, um sincronizador interno à parte operativa, ou seja, um relógio.

O autômato operacional pode ser Mealy ou Moore (/FLE 80/, /KOH 70/, /CLA 73/). Em termos de testabilidade e segurança, os últimos são certamente mais recomendáveis para implementação, visto que a função de saída N será uma aplicação $N:Q^{\alpha}\rightarrow O$, e portanto as saídas dependerão apenas do estado atual. Contudo, a opção final é sempre do projetista, pois autômatos de Mealy geralmente tem menor número de estados do que os de Moore.

Ainda segundo o modelo proposto por Davio, um programa P é uma seqüência de instruções N_0, N_1, N_2,... N_{p-1}. Cada instrução possui a forma final como a da figura 4.1, onde $f_i(x)$ simboliza funções booleanas das variáveis

condição x_i, c_i são os comandos e N_i são etiquetas para as próximas instruções.

Este modelo de programa permite um mapeamento quase imediato de construções de alto nível para a estrutura da figura 4.1. Exemplos de interpretações encontram-se na figura 4.2. Em a tem-se a interpretação para circuitos tipo Mealy, enquanto que em b o circuito é Moore. O símbolo $ significa comando vazio, enquanto que T é sempre verdadeiro.

```
N   f0(x)      c0      N0
    f1(x)      c1      N1
     ...        ...     ...
    fn-1(x)    cn-1    Nn-1
```

Fig. 4.1 - Uma Instrução segundo o Modelo de Davio

```
while f(x) do c
N1  f(x)  c   N1
      _f(x) ...  ... ;

repeat c until f(x)
N1  t     c   NJ;
NJ _f(x)  $   N1
      f(x) ...  ... ;

if  f(x)  then  c0
          else  c1
N1  f(x)  c0 ...
      _f(x)  c1 ... ;
```

a: interpretação MEALY

```
while f(x) do c
N1  f(x)  $   NJ
      _f(x)  $   Nk ;
NJ  t     c   N1 ;
Nk  ...   ...  ... ;

repeat c until f(x)
N1  t     c   NJ;
NJ _f(x)  $   N1
      f(x) ...  ... ;

if  f(x)  then  c0
          else  c1
N1  f(x)  $   NJ
      _f(x)  $   Nk;
NJ  t     c0 ...;
Nk  t     c1 ...;
```

b: interpretação MOORE

Fig. 4.2 - Exemplos de Interpretação

Na figura 4.3 encontra-se um dos possíveis

modelos de autômato de comando e autômato operacional capazes de implementar programas como anteriormente definidos. X é o conjunto de sinais de condição (x_i), enquanto que Z é o conjunto de sinais de comando (c_i). Note-se que o autômato operacional é Moore, enquanto que o de comandos é Mealy.

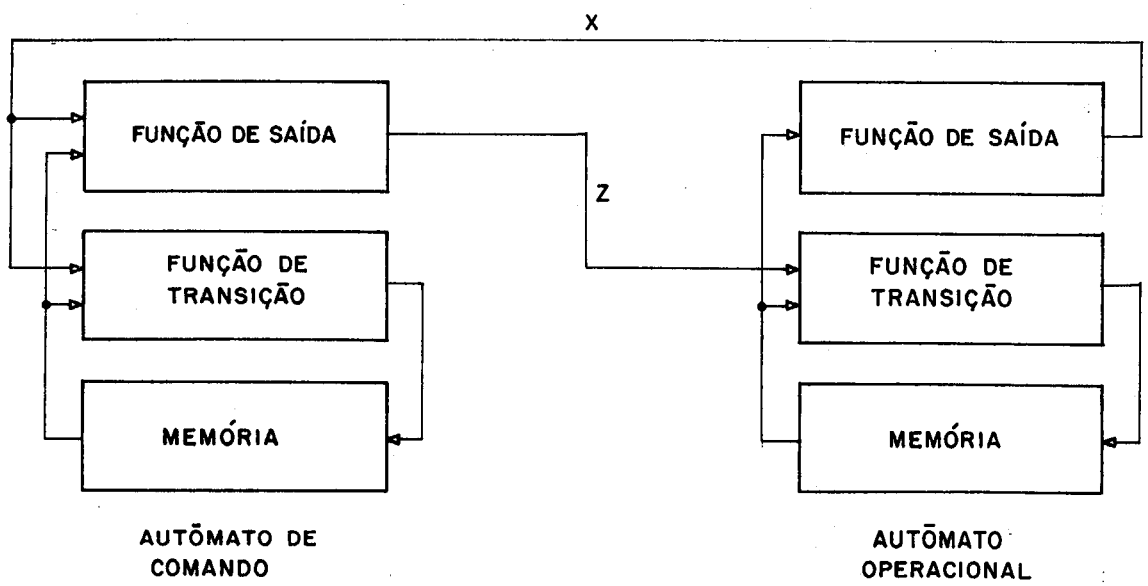


Fig. 4.3 - O Modelo para o Interpretador

Os comandos c podem disparar funções extremamente complexas no autômato operacional, a ponto deste ter de ser novamente decomposto em PC-PO recursivamente como mencionado no capítulo 3.

4.1 O Modelo da Parte Operativa Base

Para partes operativas base (isto é, que não serão decompostas em sub-partes de comando e operacional) adota-se um modelo um pouco diferente do apresentado, mas ainda contido neste.

Segundo Suzim (/SUZ 88/), uma parte operativa é uma tupla {M,O,C,Co,St,I/O}, onde se representa um conjunto

de elementos de memória, de operadores, de recursos de conexão, de comandos, de informações de estado e de portas de entrada e saída respectivamente.

Um subconjunto de comandos não conflitantes direcionam a parte operativa para executar uma ou mais ações. O número destas que pode ser feito em paralelo por uma parte operativa é o seu grau intrínseco de paralelismo. Pode-se ter mais de uma parte operativa trabalhando sob comando de um mesmo autômato de controle para aumento de paralelismo na execução de um algoritmo, conforme exemplificado no capítulo anterior.

Este modelo é extremamente útil quando se pensa na implementação física da parte operativa base. Isto se explica pelo fato de que o número de estados significativos para o controle de uma PO base é reduzido diante do número total. Levar em consideração todos os estados possíveis nos elementos de memória permite um conjunto de estados gigantesco para as partes operativas usuais nos circuitos integrados

Exemplifica-se com o seguinte problema: seja uma PO com 16 registradores de 16 bits e alguns operadores quaisquer. Contando-se uma alteração em um bit como troca de estado, verifica-se que seriam passíveis de descrição 2^{256} estados!

Muitas vezes, somente alguns bits dos registradores e alguns resultados de operações (carry, overflow, negativo, zero, informações de status em geral) interessam ao autômato de controle. A função de transição limita-se geralmente a selecionar caminhos e operações sobre os registradores. Portanto, as PO base são autômatos dos quais apenas um pequeno subconjunto de estados é realmente significativo para o controle.

Por fim, é necessário considerar que devem existir caminhos (C) entre os registradores e operadores, além de pontos de entrada e saída externos à parte

operativa (I/O).

4.2 A Automatização da Concepção da PO

As grandes dificuldades além do layout para implementação de partes operativas base encontram-se na correta manipulação dos elementos de memória, operadores e recursos de conexão. Pode-se dizer que a concepção de PO base pressupõe uma série de compromisos físicos (área, freqüência, consumo) e de otimizações (graus de liberdade sobre os compromissos físicos), envolvendo o número e tipo de memórias, número e tipo de recursos combinacionais (operadores), a complexidade da conexão e o tempo disponível para cálculo de funções.

Um estudo interessante para exemplificar o parágrafo acima encontra-se em /GRA 83/, onde se analisam diferentes implementações no nível RT da seguinte descrição ISPS:

```
BEGIN
**Carriers**  /* declarações */
t1 <0:15>
t2 <0:15>
a  <0:15>
b  <0:15>
**Activity**
action :=
(t1<-  a+b  next  /*  lista de operações */
 t2<- a-b next    /* sequenciais */
 a <- t1+t2 next
 b <- t1-t2 )
END
```

Comparam-se 5 implementações a nível de registradores e operadores para implementação do comportamento acima descrito. A implementação em silício foi feita com o uso de uma biblioteca CMOS de standard cells. Algumas avaliações anteriores à implementação em silício não corresponderam à realidade do circuito construído pela pouca ligação entre o

programa de síntese automática e o método de implementação em silício. É evidente que o melhor banco de escola para se descobrir as otimizações corretas a efetuar e aquilo que se pode abstrair na concepção de um circuito é a própria estação de trabalho: é pelo acúmulo de experiências de concepção de circuitos que se realizarão as melhores ferramentas.

A automatização do processo de construção de partes operativas a partir de uma descrição comportamental é tarefa delicada e ainda objeto de estudos, como aqueles encontrados em /TSE 87/, /DAV 80/, /HIT 83/ e outros. Fundamentalmente, tenta-se chegar a uma solução através de refinamentos sucessivos, amarrando detalhes estruturais e geométricos à arquitetura que se deseja atingir. Estas amarras são fornecidas pelo projetista através de restrições como freqüência e área, ou através da explicitação de paralelismo. Os problemas principais que um sistema de implementação automática a partir de descrições comportamentais encontra são a alocação de estados e os recursos a serem utilizados. Por alocação de estados entende-se aqui o estabelecimento do sequenciamento das ações, envolvendo o estudo de critérios para a execução serializada de ações que poderiam ser realizadas em paralelo. A decisão influencia a parte operativa no número e tipo de recursos utilizados, além de naturalmente trazer conseqüências de área, consumo e freqüência.

Concomitantemente ao processo de escolha do número de estados (serialização/paralelização do algoritmo), deve-se ter um processo para descobrir que funções (operadores) estarão presentes na parte operativa, visto que estas podem ser compartilhadas. É preciso também decidir quantos registradores correspondem às variáveis, visto que nem todas são utilizadas ao mesmo tempo durante o desenrolar do algoritmo. Nesta mesma fase é importante decidir os meios de comunicação que serão utilizados entre os registradores e os operadores, avaliando-se o custo que

representam para diferentes implementações.

Além destas opções arquiteturais, deve-se levar em conta a estratégia de sincronização (tipo de relógio), mono ou polifásica, o que leva a grande possibilidades de alterações físicas de registradores e operadores em busca de melhores figuras de mérito em relação a área, freqüência e consumo. Achar o ponto ótimo de todas estas variáveis no plano de concepção é tarefa atualmente baseada na heurística, dependendo muito da experiência dos projetistas envolvidos.

Para facilitar o processo de concepção automática, várias técnicas podem ser utilizadas. A linguagem de entrada pode explicitar paralelismo e o número de registradores, o tipo de recurso de comunicação a ser utilizado (barramento ou multiplexador), assim como os operadores (ULA, somadores, contadores, etc.)

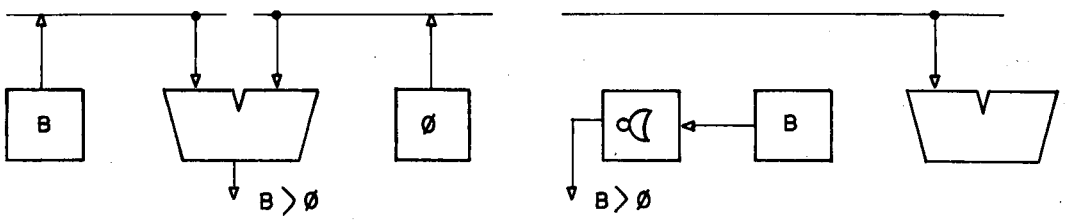


Fig. 4.4 - Diferentes implementações de If B>0 Then ...

Mesmo com estes auxílios do projetista humano, o programa interpretador para passagem do nível algorítmico para um nível mais abaixo tem muitas decisões a tomar. Por exemplo, a operação "If B>0 Then ..." pode ser implementada com uma passagem pelo registrador B pela ULA (fazendo a comparação com zero), ou pode ser feita automaticamente pelo uso de um operador NOR a saída do registrador B, como na figura 4.4, desde que B seja um número positivo.

A inclusão deste novo operador liberaria a ULA e o caminho até ela para o uso de um outro registrador em outra operação, com o custo adicional do novo operador NOR.

Claro está que decisões arquiteturais e de sequencialização de estados são função do número de registradores, dos operadores e da conexão, além de sua forma, área e potência. Mesmo com o uso do HAD para colocação de restrições, o programa interpretador terá muito a realizar para passar do nível algorítmico para o nível de máscaras.

4.3 O Modelo da PO Base para Concepção Automática.

Sistemas de implementação tradicionais adotam procedimentos nos quais o modelo comportamental do sistema é traduzido em estrutura através de uma seqüência de transformações e otimizações (/GAJ 86/). Apesar de geral e flexível, corre-se o risco de se perder a habilidade para incorporar a informação concernente ao silício. Isto se refletirá em perda de área e desempenho. Além disto, a generalidade do enfoque acarreta uma grande dimensão do espaço de concepção a ser explorado.

Colocar à disposição de ferramentas de implementação automática todas as possibilidades de se realizar registradores e operadores é tarefa no mínimo extremamente ambiciosa. Para diminuir o espaço de projeto e permitir otimizações dedicadas, imagina-se uma classe de interpretadores capaz de traduzir descrições algorítmicas em silício para uma arquitetura alvo especializada, qual seja, a de processadores monolíticos (/SUZ 81/).

A arquitetura, sendo fixa, limita o espectro de aplicação do interpretador, mas permite atingir-se um circuito ótimo e próximo do obtido pelo projetista humano, visto que este é o poder da especialização. As técnicas de otimização podem ser agora heurísticas, pois dedica-se atenção a propriedades da arquitetura alvo e ao silício a ela associado.

Os processadores são sistemas digitais programáveis, e como tal não são adaptados para todos os casos de sistemas digitais, mas para uma grande e significativa maioria. Tendo-se uma parte operativa base, uma simples reordenação de comandos permite a execução de um algoritmo diferente. Sendo assim, a escolha desta arquitetura alvo traz o poder da especialização (circuitos otimizados em área e desempenho) com o poder da generalidade (circuitos digitais programáveis).

Para otimizar a construção em silício dos elementos que compõem a PO base, limita-se a entrada do gerador a apenas a componente estrutural do circuito com um conjunto de parâmetros. O usuário, seja ele humano ou um programa interpretador, especificará as operações que deseja, o número e tipo de registradores, recursos de conexão, o número de fases do relógio, as técnicas de implementação de operadores e o conjunto de portas de I/O. Deixa-se portanto todas as tarefas onde é necessária a criatividade ao utilizador do gerador, colocando-se o computador a trabalhar sobre os problemas cansativos e sujeitos a muitos erros humanos como desenho das células, avaliação elétrica e topológica, montagem das células, etc.

As primitivas arquiteturais da PO base são traduzidas para um conjunto de módulos parametrizáveis, capazes de gerar, sempre para partes operativas base, um layout o mais próximo possível daquele que seria obtido por um ser humano, devolvendo ao usuário do gerador informações dependentes do silício que influenciam na implementação. Para um conjunto de primitivas arquiteturais que implementam um algoritmo, o GPO pode fornecer ao programa interpretador que o utiliza (ou ao projetista humano) informações ligadas diretamente a técnicas de implementação, perfazendo um esquema de consultas como descrito em /CAR 88a/.

4.4 Elementos Arquiteturais da PO Base.

Resta definir, para a arquitetura alvo de circuitos do tipo processador, os elementos da parte operativa base que farão parte desta. Durante a execução de algoritmo em silício, a cada etapa de cálculo existe um conjunto de operações ativo e um conjunto de registradores que fornece os dados para os operadores e recebe dados destes, como na figura 4.5

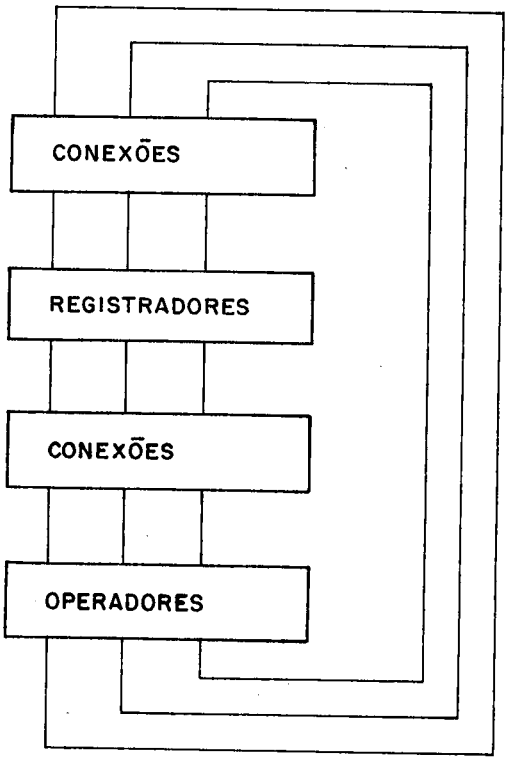


Fig. 4.5 - Elementos para execução de um algoritmo

Se existem transferências simultâneas em cada etapa, e se cada registrador pode ser ligado a outro qualquer durante certa instrução, tendo-se N entradas para o circuito de conexão deve-se possuir também N saídas. Seja M o número de bits por registrador. Será então necessário o uso de M multiplexadores de N bits para obter-se o roteamento total, de modo a possibilitar a troca de conteúdo entre todos os registradores.

O recurso de conexão utilizado (multiplexador)

permite trocar simultaneamente o conteúdo de N registradores. O custo deste circuito é proporcional à N^2, visto que se para cada N registradores tem-se um multiplexador, para se colocar mais um registrador deve-se também colocar N+1 multiplexadores, como na figura 4.6, para 3 registradores de 4 bits.

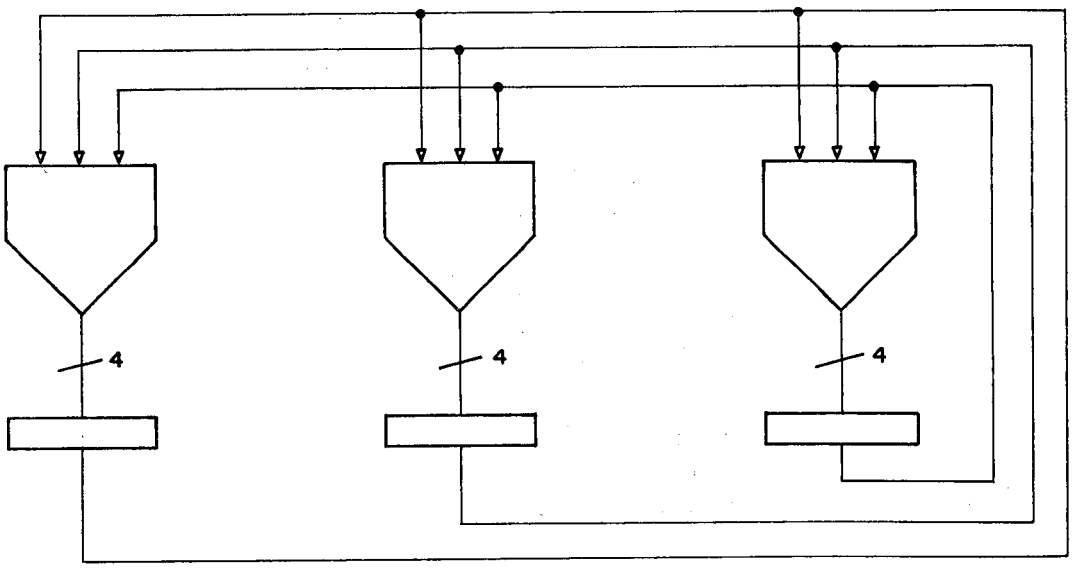


Figura 4.6 - Circuito de conexão com multiplexador

O circuito da figura 4.7 permite apenas uma conexão em cada instante. Cada registrador com saída três estados é conectado ao barramento. Embora mais lento, o custo deste circuito é proporcional à N, visto que para acréscimo de novos registradores tem-se já o multiplexador embutido.

Seja c1 o custo de um multiplexador 2:1. O custo C_{TM} da implementação dos recursos de conexão com multiplexadores será, para N registradores de M bits, igual a (M-1)*N*c1, onde M-1 significa o número de multiplexadores necessário para isolar-se um bit do conjunto de registradores (por exemplo, um dos bits zero da figura 4.6). Se c2 é o custo de uma saída três estados ao registrador para conectá-lo ao barramento, o custo C_{TB} para

uma implementação como a de figura 4.7 será de M*N*c2.

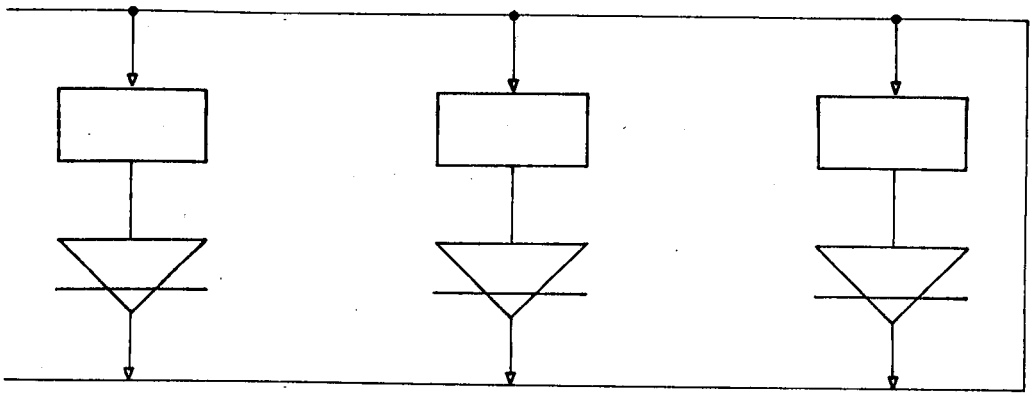


Figura 4.7 - Circuito de conexão com barramento

A aparente igualdade entre os custos é desmentida pelos seguintes fatos:

-quando do acréscimo de mais um registrador, as fórmulas passam a ser C_{TM}= (M-1+N)(N+1)c1 e C_{TB}=(M)(N+1)c2, o que comprova o acréscimo quadrático em relação a N de C_{TM};

-o custo físico de c2 é menor que c1, pois além dos transistores, deve-se levar em conta o custo da fiação de roteamento.

A discussão acima permite colocar luz sobre o fato de que uma redução sensível de custos pode ser obtida com o uso da sequencialização. É preciso se levar em conta que a grande maioria dos algoritmos não necessita de todas as transferências simultaneamente, nem se restringe a apenas uma transferência por operação. Situações intermediárias são desejáveis e devem ser estudadas. Em outras palavras, tenta-se evitar a implantação de recursos (consumindo área) que não serão utilizados. Se o algoritmo realiza uma transferência em um determinado passo, apenas uma conexão e um multiplexador são utilizados, permanecendo os demais inativos. Definem-se geralmente ações de até três variáveis (a = b op c), necessitando-se no máximo de três

conexões simultaneamente.

A realização de duas ou mais ações em paralelo pode se apresentar como:

-as variáveis são independentes, isto é, pertencem a domínios disjuntos; neste caso são dois algoritmos paralelos e duas POs podem ser construídas;

-as variáveis são independentes mas possuem necessidade esporádica de comunicação, e neste caso implementa-se um recurso de comunicação entre as POs;

-as variáveis correspondem a valores distintos no tempo, como quando fazem parte de uma estrutura pipeline.

Para a arquitetura alvo de processadores, a estratégia de conexão por barramentos é a mais adequada para implementação em silício, pelos motivos aqui expostos e aqueles encontrados a seguir.

Como já se tem definida a arquitetura alvo e o modelo de conexão mais adaptado a ela, pode-se agora atacar o problema dos elementos de memória e operadores, no sentido de se estabelecer sua estrutura e como irão iteragir para formar uma PO base.

A fim de se avaliar as diferentes possibilidades de implementação foram estudados três exemplos de circuitos, procurando-se, a partir da descrição algorítmica, extrair a PO de cada um deles e maneiras de implementá-las.

4.4.1 O MDC

Um algoritmo para cálculo do máximo divisor comum (/DAV SD/) encontra-se na figura 4.8. Na figura 4.9 tem-se o autômato de Moore equivalente. Neste existem 12 estados, enquanto que uma máquina de Mealy equivalente utilizaria apenas 5. Contudo, mesmo mantendo-se o autômato de Moore pode-se obter uma redução no número de estados. Observe-se que os estados 3 e 9 existem apenas para teste de

resultados de operações, visto que para esta primeira aproximação imagina-se que o cálculo de x!=0 será feito pela operação x+0 com subsequente verificação do bit resultado-zero da ULA. Analogamente, x>=y é implementado pela operação x-y, verificando-se os bits de positivo e zero da ULA.

Para avaliação do comando case são necessários os bits zero dos registradores x e y. Logicamente, deve existir uma comunicação (fios) entre o referido bit de cada registrador e a PC. Nada impede que, para a avaliação de x!=0, existam também fios que levem todos os bits de x para a PC. Naturalmente, esta solução não é absolutamente adequada pelo desperdício de área devido à fiação, além do significativo aumento do número das variáveis do controle, tornando o autômato muito maior do que o necessário. Uma solução mais plausível seria a colocação de uma porta NOR conectada a todos os bits do registrador. Se x tem zero em todos os seus bits, o NOR terá saída um. A operação x!=0 limita-se agora à verificação do valor da saída do NOR. Ora, com esta solução economiza-se um estado do autômato, pois a operação x+0 (que deveria ser feita no estado 2) desapareceu. Houve é verdade um acréscimo pequeno de custos (NOR), mas a máquina de controle tem o mesmo número de variáveis de entrada e um estado a menos. A decisão sobre o que economizar fica a cargo do projetista. Apesar deste tipo de aproximação parecer trivial, é preciso chamar a atenção que o uso de técnicas semelhantes pode levar a significativo aumento de paralelismo de operações em alguns algoritmos: a ULA agora encontra-se liberada, assim como o recurso de conexão. Para efeito do modelo da PO esta operação significa a inclusão de mais um operador. No caso particular, apenas a variável x tem acesso ao operador, e portanto a conexão é dedicada.

Se para eliminação do estado 2 basta a inclusão de um simples NOR, a retirada do estado 8 é mais custosa: será preciso um comparador ligado à x e y para obtenção de

x>=y. O comparador é um circuito combinacional caro, mas dependendo das limitações em relação ao número de estados pelos quais o autômato deve passar para execução do algoritmo ele pode ser válido. Levando-se em conta as otimizações aqui expostas, o autômato final encontra-se na figura 4.10.

```
1       SYSTEM MDC;
2       main()
3       {
4       LOCAL Z[16], X[16], Y[16];
5       Z=0;
6       while (x!=0)
7               { case (X[0], Y[0]) of
8                       (0,0): COBEGIN
9                               X=X/2;
10                              Y=Y/2;
11                              Z=Z+1;
12                              COEND; break;
13                      (0,1): X=X/2; break;
14                      (1,0): Y=Y/2; break;
15                      (1,1): if (X>=Y)
16                                X=X-Y; break;
17                             else
18                                Y=Y-X; break;
19              }
20      OUTPUT(Y*2^Z)
21      }
```

Fig. 4.8 - Algoritmo do MDC

O número de estados do algoritmo foi otimizado a um custo não definido, porém maior do que o inicial, devido ao acréscimo do NOR e do comparador. Deve-se agora precisar exatamente este custo, que será função da velocidade de resposta desejada para uma transição de estado. O comparador poderá ter a propagação de carry normal, ou look-ahead. Se a PO permitir, pode-se ter um circuito dinâmico, que é mais econômico em termos de área mas exige projeto lógico e elétrico mais delicado. Estas preocupações serão trabalhadas mais adiante, para que se estude ainda o problema de paralelismo de ações na PO base.

Na linha 8 do algoritmo do MDC tem-se um comando indicando três ações paralelas (x=x/2, y=y/2, z=z+1). Se estas operações forem executadas por uma ULA, seriam

necessárias três, além da conexão dos registradores às ULAs e vice-versa. A área consumida por tal circuito é enorme. Uma solução mais vantajosa (justamente pela economia de conexão) é obtida pelo uso de registradores-operadores (reg_ops). Um registrador-operador é um registrador acoplado a um circuito combinacional que opera uma função sobre o conteúdo do registrador. Sob comando da PC, o resultado da função pode ser reescrito no registrador. Assim, contadores, deslocadores à esquerda e à direita, subtradores/somadores de variáveis com constantes e comparadores são candidatos a implementação como reg_ops.

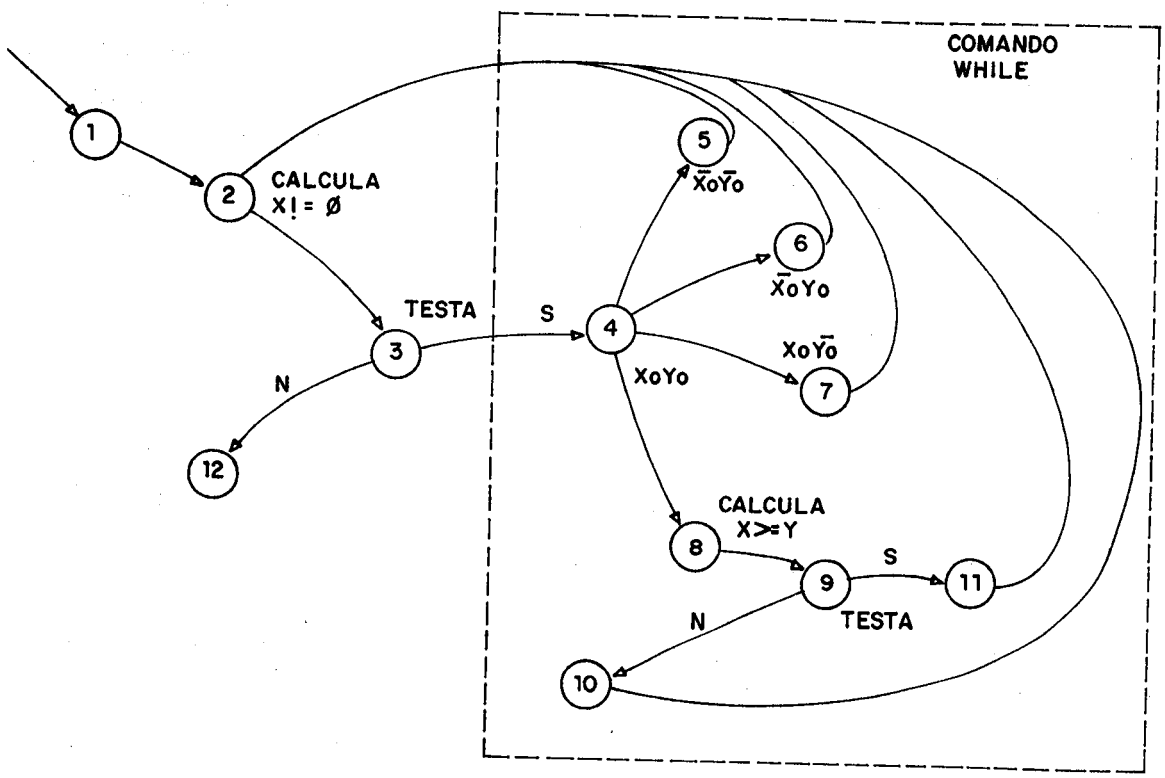


Fig. 4.9 - Autômato de Moore do MDC

Obviamente o custo em área de um reg_op é maior do que o de um registrador comum. A economia vem da conexão e do tempo de execução: como não se ocupa o barramento para realizar a função, permite-se que aquele esteja livre para comunicação entre outros módulos.

Na parte operativa do MDC existem três reg_ops, um contador e dois deslocadores à direita. Até este

instante não foi necessário nenhum recurso de conexão, visto que a parte de memória do reg_ops está sempre junto à parte combinacional. Na linha 16 do algoritmo nota-se a primeira ocorrência de uma operação diádica, x=x-y. Será necessário um subtrator, e logicamente um caminho até ele. Como na linha 17 tem-se y=y-x, o subtrator deve possuir saídas tanto para o registrador x quanto para o y. Uma primeira aproximação sugere o uso de três barramentos, como na figura 4.11. Considerando-se a subtrator puramente combinacional e os registradores possuindo apenas o comando de carga, a estrutura apresentada permite que se realize a transferência/operação de tal maneira que a um estado do autômato corresponda um estado da máquina. Pode-se ter uma economia de recursos pela criação de fatias de tempo na operação da PO. Na seção 4.6 comprovar-se-á que a simples inclusão de dois registradores auxiliares provoca uma diminuição no número de barramentos.

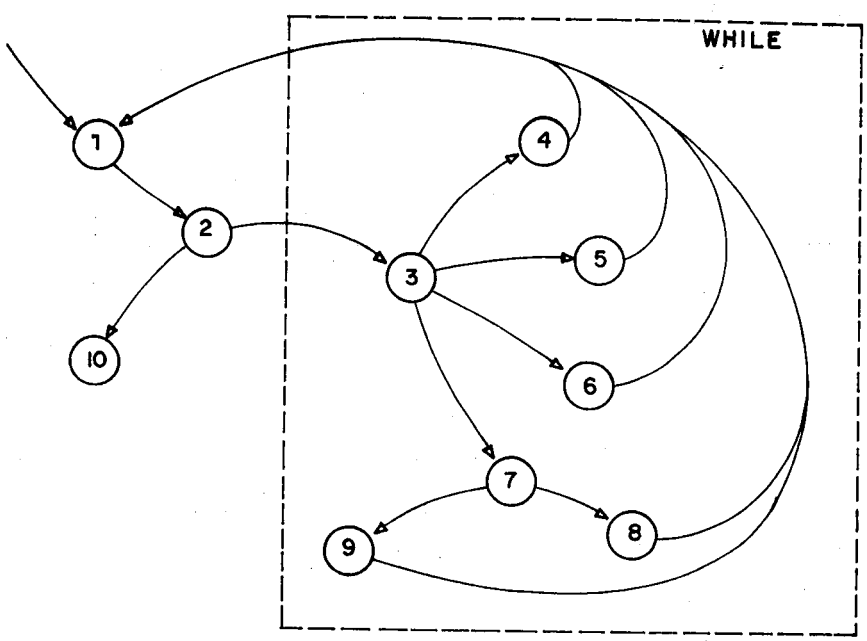


Fig. 4.10 - Autômato do MDC otimizado

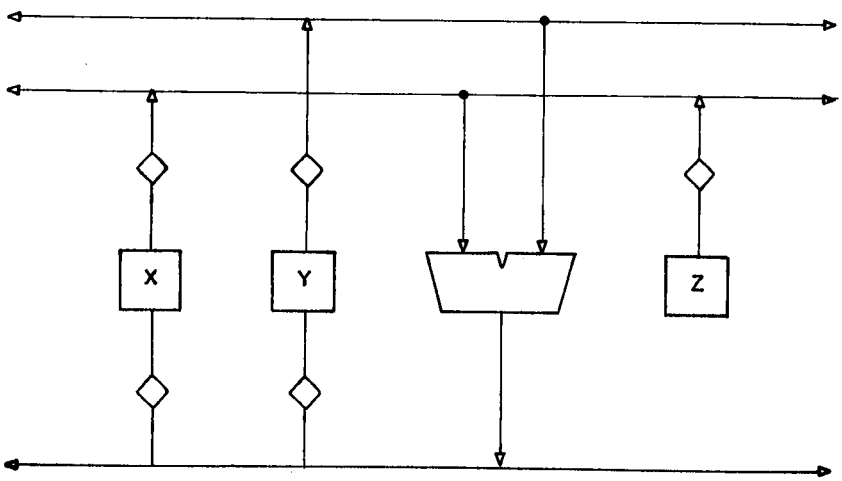


Fig. 4.11 - MDC com três barramentos

4.4.2 A Divisão Euclidiana.

A divisão de dois números inteiros pelo algoritmo de Euclides encontra-se na figura 4.12. Em princípio, a análise geral do algoritmo é idêntica à do exemplo anterior. A novidade encontra-se no registrador Q. Sobre ele são possíveis duas operações, Q=Q+1 e Q=Q*2. Poder-se-ia utilizar um reg_op deslocador, fazendo-se a operação Q=Q+1 através de um somador. O problema é que se ocuparia o barramento para transmissão de Q, inviabilizando a operação em paralelo da linha 20 (figura 4.12); o mesmo raciocínio impede o uso de apenas um reg_op contador.

Existem para este caso duas soluções imediatas. A primeira segue a linha do que foi até aqui exposto, ou seja, Q seria um reg_op contador/deslocador. O projeto de Q seria mais complexo, mas o barramento e o somador estariam livres. A outra opção seria a adoção de uma PO como em /JAM 86/, em que existe um particionamento do barramento (figura 4.13).

```
1       SYSTEM EUCLIDES;
2       main()
3       {
4       LOCAL C[16], A[16], B[16], Q[16], R[16];
5       INPUT(B);
6       C=B;
7       while (C<=A)
8               C=C*2;
9       COBEGIN
10              R=A;
11              Q=0;
12      COEND;
13      while (C>B)
14              {
15              COBEGIN
16                 Q=Q*2;
17                 C=C/2;
18              COEND;
19              if (R>=C)
20                 COBEGIN
21                      Q=Q+1;
22                      R=R-C;
23                 COEND;
24              }
25      }
```

Fig. 4.12 - Algoritmo para divisão Euclidiana

Nesta configuração, existem dois operadores complexos (um somador e um subtrator) que podem operar em paralelo. Contudo, observando-se o algoritmo, nota-se que em nenhum momento aconteceu uma operação do tipo R=f(Q) ou vice-versa. E como se a sub-parte operativa que contém Q não necessitasse da sub-parte que contém R e C. Referindo-se ao capítulo 3, a situação é a de uma PC controlando duas PO base.

Em relação à geração automática da PO, se é oferecida a possibilidade de implementação através de reg_ops, a ferramenta torna-se mais geral, sendo capaz de atacar uma gama maior de problemas: sub-partes operativas podem ser geradas como PO base e depois conectadas com chaves.

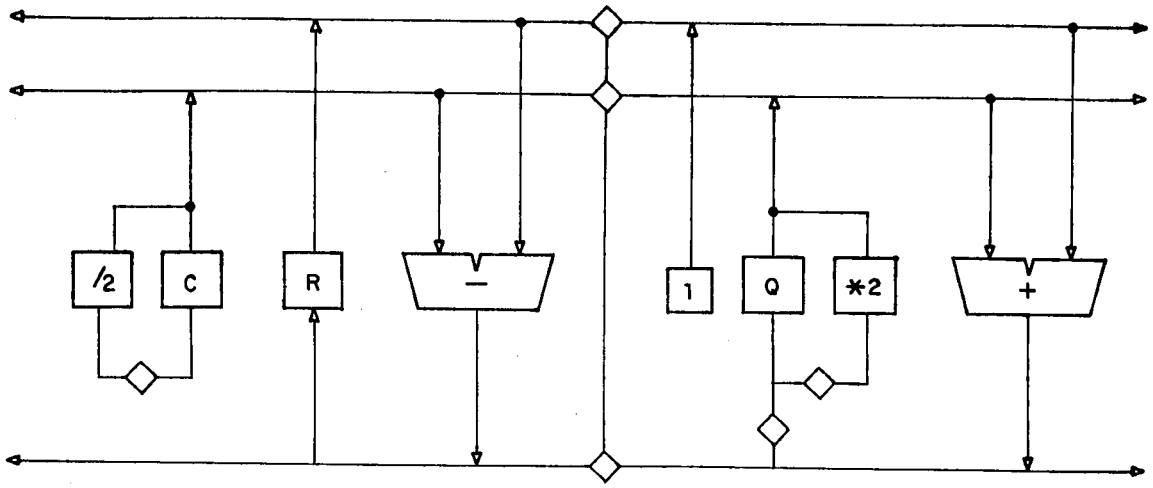


Fig 4.13 - Barramento particionado

As chaves trazem problemas de redundância e duplicação de circuitos como amplificadores e buffers de pré-carga (caso existam), devendo ser evitadas tanto quanto possível. O uso de reg_ops leva a POs com poucos operadores caros como ULAs, visto que na maioria dos algoritmos estudados as operações em paralelo são sempre do tipo x=f(z,y) em paralelo com t=g(t). A necessidade de sub-partes operativas se faz presente quando existe mais de uma operação diádica por estado, como por exemplo em x=f(x,y) em paralelo com z=g(z,t). Neste caso, a colocação de chaves no barramento é bem mais econômica que a inclusão de novos barramentos para conexão, principalmente para um grande número de bits na PO.

A regra geral de qualquer modo é difícil de ser estabelecida: cada algoritmo poderá ter novas peculiaridades ainda não estudadas. Para uma ferramenta automática deve-se buscar a versatiliade necessária para permitir a comparação de diversas opções a implementar.

4.4.3 A Multiplicação

Seja o algoritmo de multiplicação de dois números

inteiros de 8 bits da figura 4.14. Novamente o uso de reg_ops e uma ULA (somador) soluciona o problema. A dificuldade de se encontrar uma estrutura para a PO advém do fato de que o produto de dois números inteiros de 8 bits será um número de 16 bits. Tem-se portanto dois tamanhos de registradores: 16 (A e C) e 8 bits (B). A PO poderia ter altura de 16 bits e portanto haveria um desperdício de 8 bits do registrador B. Uma outra opção de implementação seria a definição de duas sub-partes operativas, a baixa (bits 0 a 7) e a alta (bits 8 a 15), como em vários processadores comerciais (/ANC 86/).

O problema eventual do particionamento é a entrada e saída: resultados de 16 bits devem ser transmitidos para o exterior como duas palavras de 8 bits. A opção pelo particionamento pode provocar POs mais baixas e largas, tornando possível um ajuste de área para POs com um número grande de bits.

Há casos em que pelo correto uso de operadores evitam-se os desperdícios e o particionamento em sub-partes operativas. Seja o trecho abaixo em que A é um registrador de 8 bits:

```
C=0;
for(i=1;i<A;i++)
        C=C+A;
```

Ao final do laço o registrador C teria o produto de A por A. Pode-se implementar este circuito com um somador de 8 bis, o registrador A, o registrador C de 8 bits e um contador de 8 bits. Cada vez que houvesse um carry da soma de A com C, o contador seria incrementado. A PO resultante tem altura 8, não há desperdício de conexão ou de bits de registradores, mas o resultado ainda teria de ser colocado para fora da PO em dois ciclos de saída. É preciso considerar também que, se n é o número de bits, este algoritmo tem complexidade de tempo proporcional à 2^n, enquanto que o anterior era proporcional à n.

```
1       SYSTEM MULTIPLICAÇÃO;
2       main()
3       {
4       LOCAL C[16], A[16], B[8];
5       RDY = 0;
6       C = 0;
7       while (B!=0)
8               if (B[0]==1)
9                    COBEGIN
10                     C=C+A;
11                     A=A*2;
12                     B=B/2;
13                   COEND;
14              else
15                   COBEGIN
16                     A=A*2;
17                     B=B/2;
18                   COEND;
19      RDY = 1;
20      OUTPUT (C);
21      }
```

Fig. 4.14 - Algoritmo de multiplicação

4.4.4 O PCIR

Um Processador de 16 bits com Conjunto de Instruções Reduzido (PCIR) foi desenvolvido no CPGCC-UFRGS (/TOD 86/). O projeto foi realizado seguindo-se a filosofia de se executar uma instrução a cada ciclo de máquina. Além disto, existe um pipeline de dois estágios (fetch e execute), fazendo com que a ULA e o banco de registradores esteja sempre ocupado. A descrição algorítmica do PCIR encontra-se no anexo 2.

Devido à operação em pipeline, o salvamento de contexto do processador (contador de programa, registrador de instruções, registrador de status e "stack pointer") é obrigatório: existem vários registradores adicionais para cópia dos valores do ciclo atual e do próximo.

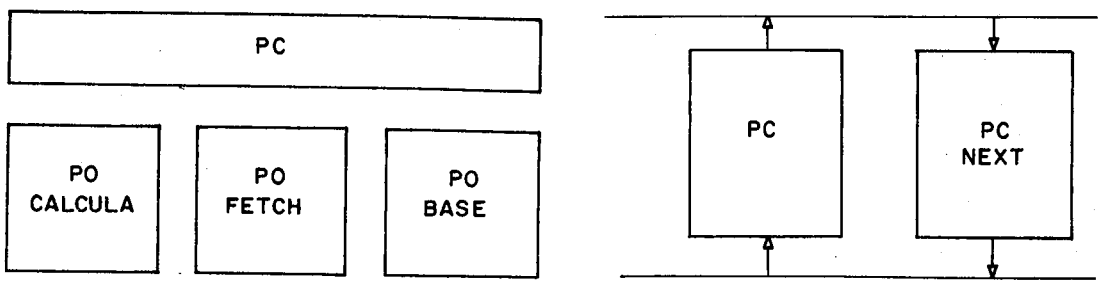
Em termos de PO, definiu-se M[x] como um operador que acessa a memória no endereço x. M[] é portanto uma função de entrada e saída. Outra função foi definida como calcula(x,y,z,t), que perfaz operações entre ULA e banco de

registradores.

Tome-se como exemplo a decodificação/execução de uma operação de soma, dada pelo seguinte trecho:

```
COBEGIN
        calcula(co,rd,rd,rf);
        RI=M[PC];
        PC=PCnext;
        PCnext++;
COEND
```

PCnext é o registrador "fantasma" de PC para manutenção do pipeline, RI o registrador de instruções. Note-se que existem 4 operações em paralelo: uma entre registradores -ULA- registradores, uma transferência de E/S, uma atribuição e um incremento. A nível de sistema digital, é interessante particionar-se o trecho acima em três sub-partes, como execução, fetch e atualização do PC. As duas primeiras exigem uma nova decomposição até se chegar a uma PO base; a última é composta por um registrador e um incrementador (figura 4.15b).



(a) (b)

Fig. 4.15 - PO do PCIR

Note-se que o registrador RI recebe o resultado de uma função cujo parâmetro é o contador de programa, ao mesmo tempo em que o próprio PC é modificado. Se RI e PC estão em POs separadas como na figura 4.15, deve-se estabelecer como conectá-las. No caso, tendo em vista que

são do mesmo tamanho, basta uma chave para que a conexão funcione. A comunicação entre os dois sistema digitais é feita através da PO.

A nível de PO base, cada um dos três sistemas do PCIR não apresenta grandes novidades em relação aos exemplos já estudados, a não ser pelo fato de existirem agora não registradores acessados isoladamente, mas agrupados em um banco, como uma memória interna à PO e conectada diretamente ao barramento. A economia de área e a redução do número de linhas de comando são grandes vantagens desta aproximação. Podem-se utilizar células de 6 transistores, e ao invés de 16 comandos para seleção de 16 registradores bastam 4. O problema decorrente é uma serialização forçada; uma operação diádica e uma monádica em paralelo somente são possíveis com um aumento signifivativo na complexidade do banco de registradores. Seriam necessários registradores com três saídas e dupla entrada, encarecendo bastante a PO.

4.5 O Relógio na PO.

É preciso dar a atenção para o conceito de sincronização. Um relógio é uma máquina que não tem entradas, apenas saídas. Seu uso dentro de sistemas síncronos é o de isolar os parasitas de tempo, transformando máquinas reais em máquinas lógicas em um dado intervalo de tempo. A máquina fica então independente ou insensível aos tempos físicos (atrasos).

O que conta para uma máquina durante a execução de um algoritmo é a evolução de estados. Esta evolução é comandada pelo relógio, com uma ou mais fases. Relógios polifásicos não carregam em si informação nova, mas apenas criam mais fatias de tempo dentro de um estado da máquina, de maneira a facilitar a implementação física da mesma.

Em todos os exemplos estudados existiam operações do tipo x=x+y; o fato do registrador x aparecer à esquerda

e à direita do sinal de atribuição provoca a necessidade do suprimento de uma barreira temporal. Esta pode ser implementada de diversas maneiras, como pela inclusão de um registrador à saída da ULA, uso de registradores mestre-escravo sensíveis à borda, etc. Dentre as mais econômicas para circuitos MOS está o uso de duas fases sem superposição.

O mais importante de um relógio (do ponto de vista da PO) é o número de fases necessário para que o autômato operacional troque de estado. Durante um estado do autômato de controle o perfil de comandos Co permanece constante, enquanto o circuito de interface valida os comandos internamente para uso da PO através das fases do relógio. Portanto, a cada estado do controle corresponde um estado da PO.

Existem, dependendo dos recursos utilizados na PO, possibilidades de uso de uma, duas, três, quatro ou mais fases, com resultante implicação em custo e complexidade (/BUD 81/, /PAR 88/, /SHO 88/). Quanto mais fases tiver um relógio, mais difícil será a manutenção e distribuição dos sinais ao longo do circuito integrado com baixo "skew" e alta freqüência. Porém, é preciso levar em consideração que circuitos polifásicos geralmente podem ser implementados com menor número de transistores, permitindo circuitos mais compactos. Cabe agora uma análise sobre os circuitos da PO base englobando-se o projeto de transistores em função do número de fases disponíveis, da quantidade dos recursos de conexão e dos compromissos entre área, velocidade e consumo.

4.6 O Mapa de Partes Operativas.

Todos os exemplos do item 4.4 foram estudados até o nível de transistores. Mesmo quando o paralelismo implícito (número de ações possíveis em paralelo) de uma PO base já se encontrava definido, ainda assim havia a

possibilidade de se obter ganho de recursos como área ou velocidade. Confrontando-se as opções possíveis, mas não com a pretensão de exaustão de alternativas, foi montado um mapa de partes operativas, que se encontra no anexo 3. O mapa possibilita comparações quanto a numero de barramentos, numero de fases, tipo de registradores e tipo de operadores.

Sobre o mapa de POs é que foi montado o gerador de POs. Ali se observa claramente a enorme variedade de parâmetros estruturais e de composição que pode haver para implementação de um mesmo circuito. Por exemplo, o algoritmo do MDC poderia ser implementado de todas as diferentes maneiras ali expostas. Uma amostra de como a implementação de um circuito pode passar por diversos compromissos de área e velocidade encontra-se a seguir.

O circuito da figura 4.16 implementa a operação x=x-y em paralelo com z=z+1. A ULA é puramente combinacional, e o diagrama de tempos mostra que os comandos de seleção podem estar ativos em Phi1, enquanto que os de carga só podem estar ativos durante Phi2. Ora, se a ULA é puramente combinacional, quando da carga de x a barreira temporal deixará de existir, e o resultado não mais será válido.

Barreiras temporais podem ser implementadas através da técnica mestre-escravo ou por circuitos sensíveis à transição (assíncronos). Portanto, x deve ser um registrador tipo mestre-escravo, como o dinâmico definido no mapa (anexo 3). Observe-se que o custo destes registradores é alto (9 transistores no mínimo), justamente devido à necessidade da barreira temporal dentro do registrador. Desloca-se aquela para a ULA pela colocação de registradores à sua entrada. Para a mesma temporização verifica-se que agora se podem utilizar registradores tipo estático simples ou dinâmico simples, que são mais econômicos. Para mais de dois registradores na PO este tipo de troca é altamenta recomendável.

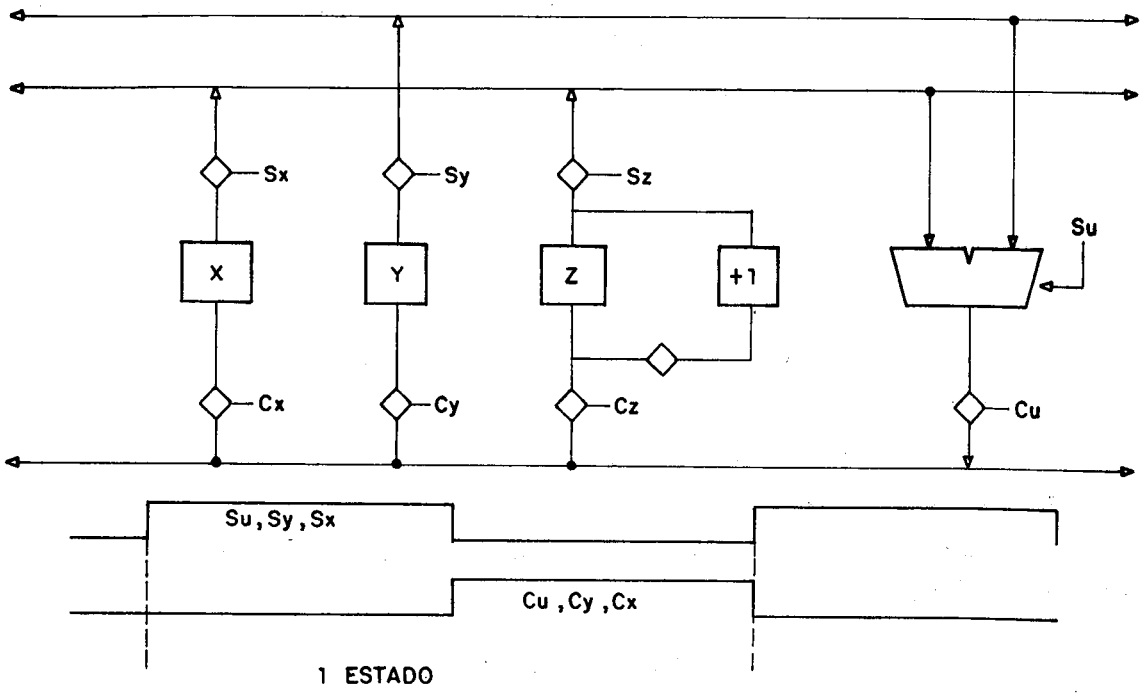


Fig. 4.16 - X = X-Y e Z = Z+1
circuito e temporização

A inclusão da barreira temporal na ULA e não nos registradores torna possível economizar um outro item, qual seja, o número de barramentos. Já que os dados encontram-se armazenados, os barramentos de entrada estão liberados, e o retorno pode ser feito por um deles, dispensando-se assim o terceiro barramento.

O circuito com duas fases, dois barramentos e registradores à entrada da ULA parece ter uma boa resposta em termos de área e desempenho. Contudo, pode-se torná-lo mais reduzido ao preço de velocidade. Com o registrador da figura 4.17, os comandos S ou C só podem ocorrer síncronos com Phi2, visto que a fase Phi1 é utilizada para realimentação. Para uma operação do tipo x=x-y serão necessários dois ciclos Phi1/Phi2. Perde-se um ciclo em relação ao circuito anterior, ganhando-se em área.

Se fosse necessária uma economia ainda maior de área, diminui-se para apenas um o número de barramentos. Neste caso, serão necessários três ciclos de Phi1/Phi2 para se completar a operação: ula_1=x, ula_2=y, x=saída_ula. A um estado do controle tem-se três ciclos PHI1/Phi2, contra uma correspondência um a um no primeiro caso estudado. A título de ilustração, na figura 4.18 tem-se duas células de registradores, uma para cada caso acima (1 e 3 ciclos). A diferença de área é evidente, assim como o número de ciclos. A decisão sobre qual opção seguir é função das especificações do projeto.

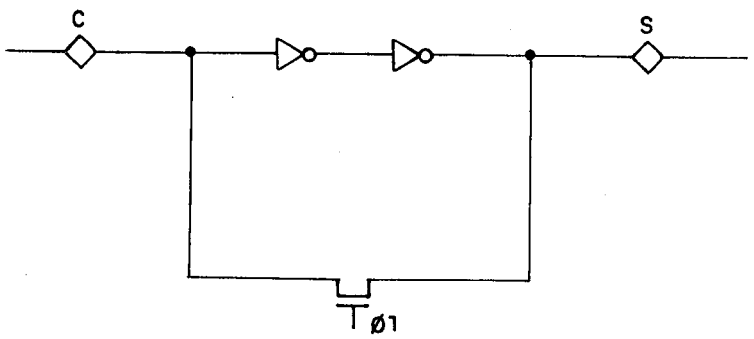


Fig. 4.17 - registrador semi-estático

Até aqui foram considerados circuitos em que o barramento não possuia pré-carga. Para POs com grande número de elementos, nas quais por conseqüência o barramento é grande, a opção do uso de um circuito de pré-carga e de buffers de descarga torna-se interessante, pois permite diminuir a área das células conectadas ao barramento. A tarefa de carga/descarga do grande capacitor fica sob responsabilidade de dois circuitos, e não de todos. Além disto, com a pré-carga os transistores que conectam as células ao barramento podem ser apenas tipo N, não sendo necessário seu complementar P. Novamente economiza-se área.

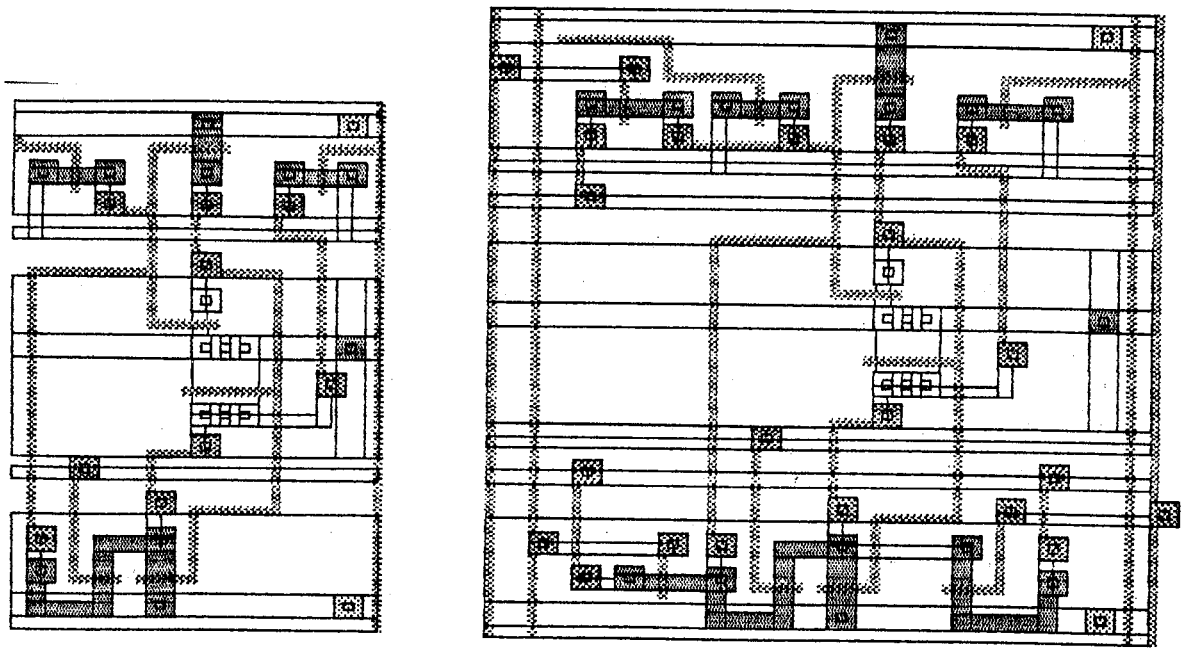


Fig. 4.18 - Registradores para um ou três ciclos de Phi1, Phi2

O uso da pré-carga acarreta que se tenha uma fase apenas para colocar o barramento em nível alto, mas pode-se contornar esta aparente perda de tempo com uma sincronização adequada com as outras operações, aproveitando tempo livre da ULA por exemplo (/TOD 86/). Não se pode descartar a hipotése de uma pré-carga assíncrona (cuja referência de tempo é o início de uma fase, e é encerrada quanda da carga do barramento, ainda dentro da fase), como descrito no mapa de POs, onde se economizam fases ao preço de testabilidade.

Finalmente, se a opção for o uso de um banco de registradores pode-se atingir grande economia de área, mas eventualmente serão necessários circuitos com quatro fases para leitura ou escrita na memória. No MC68000 por exemplo, precisam-se dois ciclos de quatro fases para uma operação do tipo Rn=Rn+Rm.

Analisando-se o mapa em relação ao gerador de POs, evidentemente é ideal que todas as possibilidades ali descritas sejam passíveis de implementação pela ferramenta. A versatilidade alcançada permitiria a construção de circuitos tão simples (em relação ao número de fases e

tipos de registradores) quanto o MDC ou tão complexos quanto processadores comerciais.

4.7 Comentário sobre a Parte de Controle.

O autômato de controle da PO base pode ser implementado de diversas maneiras. A cada uma correpondem custos em termos de área, freqüência, potência, além de atributos como facilidade de implementação, alteração e depuração.

Uma primeira aproximação seria o uso de flip-flops como memória e um circuito combinacional para implementação das funções de próximo estado e saída. Para um programa de P instruções, serão necessários n flip-flops de maneira a se obedecer $2^n >= P$. Apesar de formulação simples, é preciso considerar que para um número de estados maior que 32, o mapa de próximo estado tornar-se-á extremamente difícil de otimizar manualmente, e mesmo com o auxílio de ferramentas de CAD, é extremamente delicada a produção de um layout otimizado e automático para funções lógicas em geral. Além disto, no caso de alterações, todo o sistema deve ser refeito, o que provoca pouca flexibilidade a este modo de implementação de PCs. Contudo, existem processadores modernos que utilizam este método de implementação, como /FOR 87/ e /BER 87/.

Outra solução seria o uso direto de PLAs, que implementam mintermos de maneira razoavelmente compacta, existindo programas geradores de PLAs otimizados (/BAR 85/, /CHU 84/). Tanto a função de saída quanto a de próximo estado estão dentro do próprio PLA.

Apesar de facilmente alterável, visto que basta modificar-se a entrada do gerador de PLAs, a solução acima peca pelo fato de que, se a função de saída ou a de transição é algoritmicamente complexa, o tamanho do PLA poderá ser proibitivo, seja em termos de área, seja em termos de velocidade ou potência.

Uma sofisticação desta implementação pode ser feita através da colocação da função de saída do autômato em várias partes distintas, fazendo-se uma decomposição física da máquina de controle. Por exemplo, para n comandos c_i mutuamente exclusivos, pode-se codificá-los substituindo-os por $\log_2 n$ variáveis intermediárias, e realizando a decodificação em um bloco separado. Desta maneira, será mais fácil otimizar área e velocidade para os circuitos de controle. Uma possível implementação encontra-se na figura 4.19, onde o decodificador é uma ROM. Para alterar comandos da PO, basta alterar a linha da ROM correspondente ao estado desejado. A redução do PLA pode ser feita por técnicas de otimização lógica ou topológica (folding). Podendo-se separar as saídas do PLA em um conjunto de funções que dependam de um subconjunto das variáveis de entrada, é possível que dois ou mais PLAs constituam-se em uma opção mais eficiente do que um único PLA.

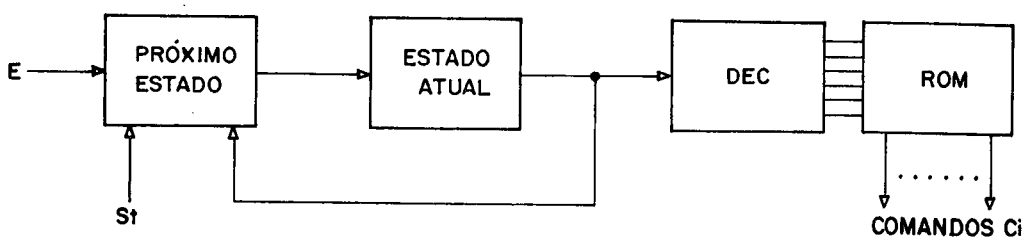


Fig. 4.19 - Parte de Controle

A otimização da Parte de Controle tem provocado diversos esforços seja na otimização lógica seja na correta alocação de estados. Para otimização de estruturas microprogramadas como acima descrito existe material relativamente abundante. Em particular, pode-se encontrar referências de microprogramação para VLSI em /DAV 83/, /ZYS 83/ e /OBR 82/. Exemplos de processadores que utilizam esta abordagem podem ser encontrados em /CHA 86/, /JOH 84/, /ANC

86/ e /BUD 81/.

Um tipo especial de programa pode ter a implementação do autômato de controle também especializada (/DAV 83/). Chamam-se programas incrementais aqueles escritos somente com o uso das três instruções abaixo ($ indica que o comando é vazio, t é sempre verdade):

-salto condicional

N_i !x $ N_{i+1}

x $ N_j;

-salto incondicional

N_i t $ N_j;

-execução

N_i t c, N_{i+1}.

Este tipo de programa traz em si uma vantagem evidente, visto que o campo de próximo estado não mais precisa existir na ROM. Pode-se substitui-lo por um contador binário com carga paralela e uma lógica combinacional para controle de saltos ou interrupções. Além disto, dispondo-se de uma pilha de memória, existe a possibilidade de economia de trechos de microprograma pelo uso de subrotinas.

Além das partes de controle microprogramadas e não microprogramadas, em /HOR 87/ descreve-se um processador onde a parte de controle é uma memória cache, ou seja, a palavra de controle é a própria instrução do processador. Neste contexto o conceito de circuito de interface tem grande influência.

Retorne-se ao item 4.6, quando a um estado do controle (que executa por exemplo x=x-y) correspondem três ciclos de duas fases na PO. Evidentemente, o relógio principal é o mesmo, tanto para a PO quanto para a PC, visto que ambas devem trabalhar sincronamente. Dependendo da implementação de cada uma é que será necessário o desdobramento do relógio em fases. Não é obrigatório que o número de fases da PC seja idêntico ao da PO. Entre ambas deve existir portanto um circuito de interface que faça o

casamento entre os diferentes sinais (figura 4.20).

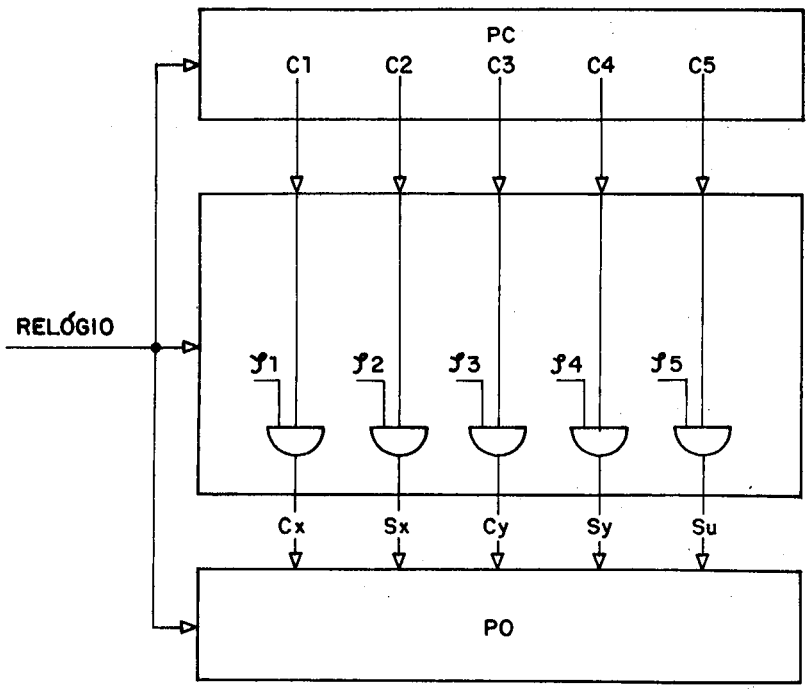


Fig. 4.20 - Circuito de Interface

O circuito de interface será tão mais complexo quanto maior for a diferença de fases entre a PO e a PC. Reciprocamente, no limite o circuito de interface será totalmente desnecessário. Na maioria dos casos, por razões práticas, no circuito de interface mesclam-se as funções de ativação dos comandos e amplificação. A primeira refere-se ao que se comentou acima, ou seja, a compatibilização entre a geração das micro-ordens para a parte operativa e os correspondentes sinais de controle. A segunda incumbe-se de resolver o problema sob o ponto de vista elétrico, pois é no circuito de interface que se encontram os amplificadores de comandos.

Os circuitos de interface podem desempenhar ainda funções lógicas quando recebem parâmetros ou padrões que devem ser interpretados. Um PLA para transcodificar o código de operação da ULA ou um decodificador de endereços

do banco de registradores podem ser considerados circuitos de interface.

5 O GERADOR DE MODULOS

O gerador de partes operativas é um programa que aceita como entrada uma descrição da estrutura da PO e gera na saída o layout correspondente. O GPO perfaz portanto um caminho entre um nível de abstração mais alto para um nível de abstração baixo, passando por sobre diversos outros, como por exemplo, o lógico. As características do gerador de POs serão aqui descritas.

Dentro de uma filosofia de projeto de ferramentas de CAD (/SUZ 85/, /CAR 88a/), três ítens são fundamentais em um gerador automático, que são a independência da tecnologia, a parametrização e a avaliação elétrica. Um gerador de módulos é caro para se construir (em termos de tempo), e portanto é impensável que a cada dois anos ou menos tenha-se de reescrevê-lo na totalidade devido a uma alteração na tecnologia. Além disto, a possibilidade dada de variação da fundição de silício (second-source) encoraja que se possa migrar facilmente de uma tecnologia a outra. Quando o termo independência de tecnologia é referenciado, sua abrangência é limitada. Um gerador de módulos CMOS com um nível de metal pode gerar máscaras para diversos processos CMOS com um nível de metal, mas não para nMOS ou bipolar sem severas alterações. Existe a independência da tecnologia dentro de uma família tecnológica.

Um módulo deve ser uma ferramenta específica para que se possam gerar layouts eficientes; ao mesmo tempo, deve-se poder utilizar o módulo em mais de um projeto. Desta contradição surge o conceito de parâmetrização. Cada vez que o gerador de módulos é chamado um conjunto diferente de parâmetros é levado em consideração. São estes que permitem um ajuste fino do módulo para o circuito em que ele deve atuar. Os parâmetros são a personalização do módulo, e portanto são os elementos que conferem as vantagens da generalidade a uma ferramenta específica.

Por fim, a avaliação elétrica é fundamental pelo

fato de que a simulação de um layout com muitos transistores é impensável em termos práticos. Contudo, a realimentação elétrica deve ser fornecida ao projetista, para que se possa avaliar o desempenho do circuito em relação às especificações do projeto, eventualmente alterando-se o conjunto de entrada do gerador para melhorar algum item não satisfatório. Portanto, um gerador deve possuir modelos equivalentes e algoritmos para verificação de atraso ou potência, permitindo ao usuário da ferramenta um rápido esquema de consultas.

O gerador de POs busca atingir todos os itens comentados. A seguir apresentar-se-ão as soluções encontradas para implementação destas características no gerador.

5.1 A Independência da Tecnologia.

A tecnologia escolhida para implementação das células foi CMOS com um ou dois níveis de metal. A decisão baseou-se na disponibilidade da tecnologia através de circuitos MPC, no baixo consumo e por ser a tecnologia utilizada atualmente para uma grande gama de circuitos, existindo diversos fornecedores com diversas regras de projeto.

Para diferentes processos CMOS existem regras comuns, como tamanho mínimo de transistor, espaçamento entre camadas, tamanho do contato, etc. O processo de fabricação sendo n-well, p-well ou twin-tub provoca também mudanças relativas ao desenho de poços e contatos de substrato. Sendo assim, gerar diretamente retângulos certamente causaria uma dependência do gerador em relação às regras da fundição escolhida.

Pode-se iniciar a independência da tecnologia do gerador pela expressão dos retângulos não em tamanho físico real, mas sim em múltiplos de um tamanho parametrizável. Por exemplo, um fio de metal mais largo poderia ser

expresso como "três vezes a largura mínima de um metal", e a distância entre camadas também seria um parâmetro.

Com esta aproximação obtém-se a independência da tecnologia, e dois geradores do GME foram assim implementados, o gerador de ROMs e o gerador de PLAs (/CAR 88b/, /HES 89/). Para layouts regulares e repetitivos este tipo de geração é bastante eficiente. Problemas advêm quando se desejam layouts irregulares: mais de uma regra de separação entre camadas pode existir. No exemplo da figura 5.1, a distância do transistor para a camada de difusão adjacente pode ser determinada por duas regras: R1+R2 ou R3, dependendo qual for a maior. O construtor do gerador de módulos deverá estar atento para que, nestes casos, na construção do layout a distância seja expressa por MAX(R1+R2,R3), onde MAX é uma função que retorna o maior de seus argumentos.

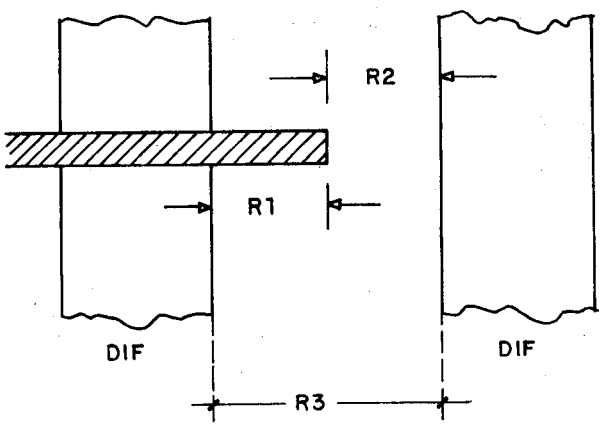


Fig. 5.1 - Regras "perigosas"

Soluções similares a esta encontram-se nos sistemas SKILL (/WOO 86/), ALLENDE (/MAT 85/), ICPL (/LEW 86/ e CONCORDE (/COR 88/). As coordenadas do circuito (a menos da primeira) são expressas sempre em posições relativas com o uso de múltiplos de regras mínimas. Entre as vantagens destaca-se o fato da linguagem ser iterativa, permitindo ao usuário toda a potencialidade de uma

linguagem de programação (Pascal, C ou Lisp).

Teoricamente, o uso de um sistema como os acima tornaria a introdução de parâmetros algo muito simples. Se uma célula é uma rotina, ela pode receber parâmetros como posição das entradas, saídas, tamanho de transistores, etc. Além disto, ela pode ser chamada por outras células, perfazendo uma hierarquia de construção de layout.

Outra solução para independência de tecnologia contempla o uso de editores simbólicos. Já não se trabalham mais com retângulos, mas sim com símbolos como transistores, fios e contatos. Cria-se um novo nível de abstração, o nível da linguagem LDS (linguagem de descrição simbólica), que se encontra acima do nível RS ou CIF (descrição de máscaras). A introdução deste nível foi necessária para independência da tecnologia.

A grande vantagem do uso de uma descrição simbólica advém da duplicidade de representação, podendo-se ter entrada gráfica ou textual. A parte gráfica é mais confortável para o projetista, enquanto que a parte textual adapta-se bem ao trabalho junto ao computador e à introdução de parâmetros. Na figura 5.2 tem-se uma célula simbólica (inversor), estando sua correspondente descrição LDS na figura 5.3.

Existem atualmente diversos sistemas simbólicos, e comparações entre eles podem ser encontradas em /EIC 86/ ou /PRE 88/. Como na época do desenvolvimento deste trabalho não estavam disponíveis editores simbólicos no GME, decidiu-se implementar um editor e montador simbólico, que serviria como base para o gerador de partes operativas.

Embora inspirado no sistema MULGA (/WES 81/), o editor possui algoritmos e estruturas de dados próprias, tendo-se obtido resultados excelentes em termos de velocidade de expansão e densidade do circuito obtido. A descrição dos algoritmos e detalhes de implementação

encontram-se em /MAR 89/.

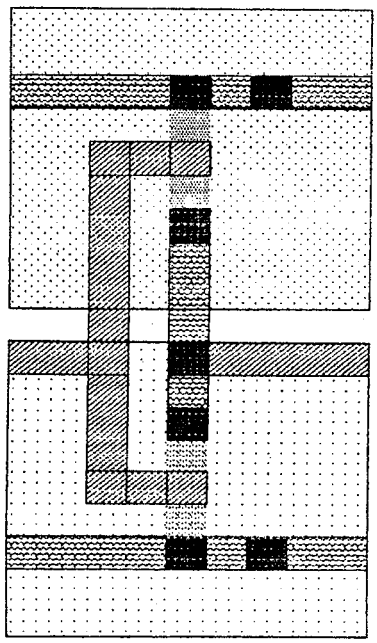


Fig. 5.2 - Inversor simbólico

Além da vantagem gráfica, o uso de um editor simbólico parece ser superior ao de uma linguagem procedural para implementação de células pela maior abstração do tipo de dado. Como se trabalham com elementos mais próximos do nível elétrico (transistores, fios, contatos), trabalha-se com um número menor de dados. Um arquivo com 40 linhas simbólicas pode gerar até 120 retângulos. O tipo de linguagem facilita o trabalho de extratores para SPICE, ARAMOS ou SLOT (simulador de chaves).

Em termos de geração de módulos, é importante verificar que a linguagem simbólica não é em absoluto limitada em relação às linguagens de descrição procedural. Também as linguagens simbólicas aceitam parâmetros, como na figura 5.3. Fios e transistores possuem um último campo chamado fator, onde é colocado o número (real) a multiplicar pelo tamanho mínimo para se obter o tamanho

final. Assim, um transistor com fator 5.50 terá W/L igual a 5.50*Wmínimo/Lmínimo para uma dada tecnologia.

```
/* inversor.lds */

N inversor ()()
M 5,10
F m 2,3 2,4
C d 2,3
T n v 1,2,3 2 2.50
W n 0,6 5,10
W p 0,0 5,4
B p 4,1
B n 4,9
F m 0,9 4,9 2.00
F m 2,6 2,7
F p 1,8 2,8
F p 1,2 1,8
F p 1,2 2,2
F p 2,4 2,6
C p 2,6
T p v 7,8,9 2
C b 2,9
C b 2,7
C p 2,4
C d 2,1
F m 0,1 4,1 2.00
```

Fig. 5.3 - Inversor simbólico textual

Na verdade, nada impede que a geração da linguagem LDS seja feita por um programa C, com passagem de parâmetros. Isto tornaria o uso da descrição simbólica equivalente ao uso de uma descrição de SKILL ou CONCORDE. A vantagem é que não é necessaria preocupação com regras do tipo MAX(R1,R2); o próprio expansor se encarrega de transformar o conjunto de coordenadas simbólicas em coordenadas reais.

Finalmente, é preciso considerar a enorme quantidade de trabalho que uma descrição textual de células provocaria. O conjunto de rotinas do gerador de partes operativas tem aproximadamente 150Kbytes de arquivo fonte para a parte das células: seria extremamente cansativa e lenta a tarefa de gerá-las manualmente.

O editor simbólico tem a vantagem da independência da tecnologia como nos editores procedurais, tem a entrada gráfica como os projetistas estão acostumados mas teoricamente carece de uma coisa importante: a parametrização automática. Quando se trabalha simbolicamente, primeiro tem-se de desenhar, para depois verificar como e onde parametrizar.

Resta portanto conjugar a vantagem da entrada gráfica simbólica com a parametrização possível através de uma entrada textual. Para tal foi desenvolvido o ggcel, ou gerador-gerador de células. Este programa aceita um arquivo LDS como entrada e produz na saída uma rotina C que compilada pode gerar o texto LDS original, correspondente ao desenho. Os parâmetros são tratados como variáveis da rotina.

Existem parâmetros que não são puramente elétricos, como tamanho de transistor ou de fio, mas significam presença ou ausência de elementos da linguagem. Assim, modificou-se o programa **charrua** (editor gráfico simbólico) para que fosse possível colocar estes elementos condicionais também como parâmetros.

Um trecho LDS com parâmetros e o correspondente programa C gerado automaticamente encontram-se na figura 5.4 e 5.5 respectivamente. O conceito de gerador-gerador é de suma importância, pois agora o projetista de células não precisa conhecer nada da linguagem de programação a ser utilizada, basta que conheça os parâmetros para fazer uma célula em um ambiente gráfico de projeto.

Para construção do ggcel foram utilizados o compilador de compiladores yacc e o analisador léxico lex em ambiente EDIX (UNIX like). As entradas yacc e lex do ggcel encontram-se no anexo 4 e 5 respectivamente.

Durante o processo de expansão simbólica, células diferentes podem resultar em alturas e larguras diferentes. Se estas células tiverem de ser conectadas, não existe

nenhuma garantia de que as entradas ou saídas estarão nas posições reais adequadas. É preciso portanto o uso de um montador simbólico.

```
/* RSE.lds */

N RSE (Phi2N,Phi2,BusC,Cc,TG,BusB,Cb,Sb,Phi1N)(Pkn,Psp,Psn,GND,VDD,Pkp)
M 25,23
W n 0,8 25,14
W p 0,0 25,4
P m 0,1 25,1 GND=2.00
P m 0,11 25,11 VDD=2.00
T n v 1,2,3 13
T n v 1,2,3 14 Psn=1.50
T p v 9,10,11 15 Psp=3.75
T n v 20,21,22 15 1.50
T n h 16,17,18 20 Pkn=1.00
C d 15,20
P p 13,19 14,19
F p 14,8 15,8

D TG=1 [
P m 17,10 18,10
P b 18,10 18,11
T p v 11,12,13 18 Pkp=2.50
C b 18,13
P p 7,13 7,18
]
```

Fig. 5.4 - Descrição LDS

O montador recebe um conjunto de células simbólicas que devem ser conectadas. Depois da expansão de cada célula folha, ele ajusta a altura de cada ponto de entrada ou saída para que as células reais possam ser colocadas lado a lado.

Esta facilidade (referenciada na literatura como pitch-matching, /BUR 86/) torna o uso do editor simbólico bastante liberal. Pode-se desenhar uma célula simbólica com qualquer número de linhas ou colunas, que o montador encarregar-se-á dos ajustes necessarios à conexão. Na figura 5.6 observam-se três células a serem montadas e a resultante. Note-se que não interessa o número de linhas simbólicas entre dois pontos de conexão entre as células. Esta característica não é encontrada por exemplo em MULGA.

```
#include <stdio.h>
extern int busca_I();
extern float busca_F();
int I_aux;
float F_aux;
void RSE(aux)
FILE *aux;
{
fprintf(aux,"M 25,23 \n");
fprintf(aux,"W n 0,8 25,14 \n");
fprintf(aux,"W p 0,0 25,4 \n");
if((F_aux=busca_F("GND"))==0.0)
        fprintf(aux,"P m 0,1 25,1 2.000\n");
else fprintf(aux,"P m 0,1 25,1 %.3f\n",F_aux);
if((F_aux=busca_F("VDD"))==0.0)
        fprintf(aux,"P m 0,11 25,11 2.000\n");
else fprintf(aux,"P m 0,11 25,11 %.3f\n",F_aux);
fprintf(aux,"T n v 1,2,3 13 \n");
if((F_aux=busca_F("Psn"))==0.0)
        fprintf(aux,"T n v 1,2,3 14 1.500\n");
else fprintf(aux,"T n v 1,2,3 14 %.3f \n",F_aux);
if((F_aux=busca_F("Psp"))==0.0)
        fprintf(aux,"T p v 9,10,11 15 3.750\n");
else fprintf(aux,"T p v 9,10,11 15 %.3f \n",F_aux);
fprintf(aux,"T n v 20,21,22 15 1.500\n");
if((F_aux=busca_F("Pkn"))==0.0)
        fprintf(aux,"T n h 16,17,18 20 1.000\n");
else fprintf(aux,"T n h 16,17,18 20 %.3f \n",F_aux);
fprintf(aux,"C d 15,20 \n");
fprintf(aux,"P p 13,19 14,19 \n");
fprintf(aux,"F p 14,8 15,8 \n");
if((I_aux=busca_I("TG"))==1)
        {
        fprintf(aux,"P m 17,10 18,10 \n");
        fprintf(aux,"P b 18,10 18,11 \n");
        if((F_aux=busca_F("Pkp"))==0.0)
                fprintf(aux,"T p v 11,12,13 18 2.500\n");
        else fprintf(aux,"T p v 11,12,13 18 %.3f \n",F_aux);
        fprintf(aux,"C b 18,13 \n");
        fprintf(aux,"P p 7,13 7,18 \n");
        }
}
```

Fig. 5.5 - Programa C gerado

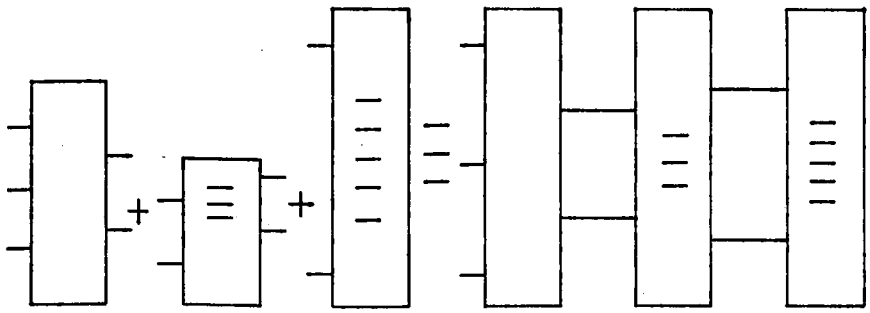


Fig 5.6 - Células a montar

Claro está que a montagem é feita pelo pior caso, ou seja, a célula mais alta é a dominante. O usuário da ferramenta pode modificar a célula mais alta de maneira a reduzir o tamanho do circuito como um todo.

O uso do montador e do ggecl possibilita a construção de módulos. É importante frisar que, até o presente momento, somente a entrada de células é gráfica. Os comandos de montagem exigem entrada textual. Em /POW 87/ e /PRE 88/ reportam-se geradores-geradores de módulos com entrada gráfica, na forma de matrizes ou diagramas de montagem.

Os sistemas mencionados exigem porém uma biblioteca de células já pronta, enquanto que a aproximação do GPO utiliza uma biblioteca de funções. Obviamente, a possibilidade de um gerador-gerador de módulos com entrada gráfica é interessante.

Outro ponto importante a ressaltar é o problema da compactação do layout. O maior defeito do uso de um editor simbólico é a necessidade de uma posterior compactação do layout. Devido ao algoritmo de expansão, pode haver desperdício de área em situações como a da figura 5.7, onde existem muitas linhas de metal com contatos. Se os contatos fossem alternados acima e abaixo haveria significativa economia de área. Contudo, para células normais, o desempenho do editor simbólico local é bastante satisfatório. Produzindo-se simbolicamente algumas células feitas por projetistas do GME obteve-se uma

perda de área ao redor de 10%.

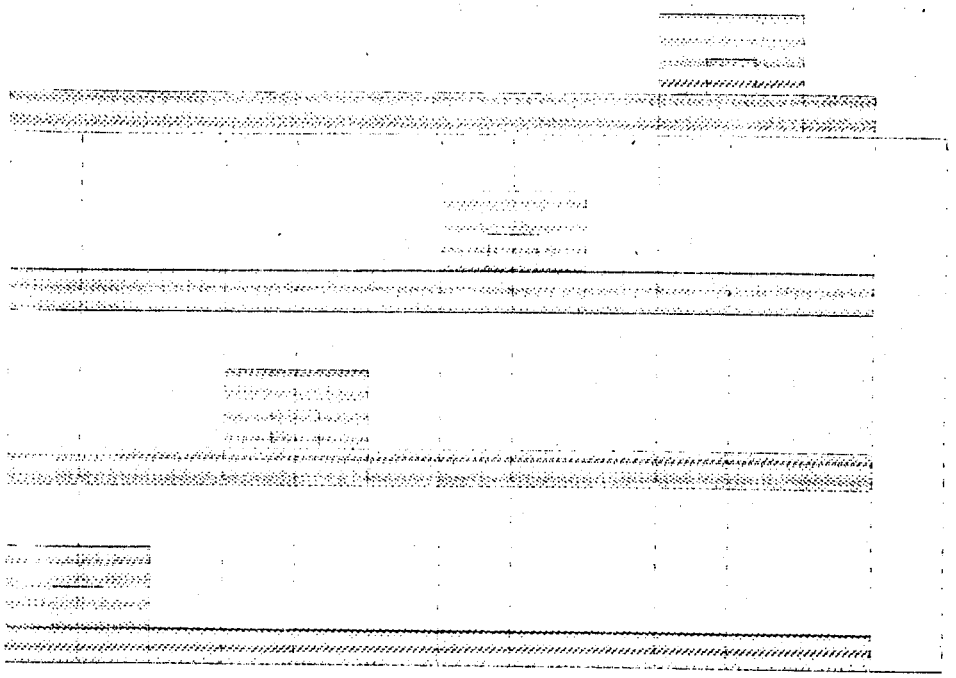


Fig. 5.7 - Contatos sem compactação

De qualquer modo, no momento em que existir um compactador para a linguagem LDS, basta executar um pós processamento sobre as células geradas para se obter um layout mais compacto. O usuário do editor simbólico pode contudo intervir decisivamente na obtenção de uma célula com pouca área: muitas vezes uma pequena mudança topológica causa um grande ganho de área (por exemplo, mudar um transistor vertical em um horizontal).

Na figura 5.8 tem-se a arquitetura dos geradores de módulos que compõem o gerador de partes operativas. Os geradores aceitam um conjunto de parâmetros, fazem uma verificação da validade destes parâmetros e depois chamam as rotinas C que geram as células componentes dos módulos em linguagem LDS. Por exemplo, a montagem de um registrador em uma PO de 16 bits é hierárquica, pois gera-se um registrador e repete-se esta célula 16 vezes, como se pode

verificar no trecho abaixo:

```
V REG = ( cell[][] ) # 16;
```

REG é um módulo com 16 instâncias verticais de cell. No gerador de módulos, a geração do código que gera células foi automatizada pelo uso de ggcel. A geração do texto do montador deve ser feita em C de maneira específica para cada caso. Note-se que a saída de cada módulo é uma descrição simbólica da célula e um ou mais comandos para o montador simbólico. No exemplo acima, cell é uma célula simbólica, enquanto que REG é um módulo simbólico.

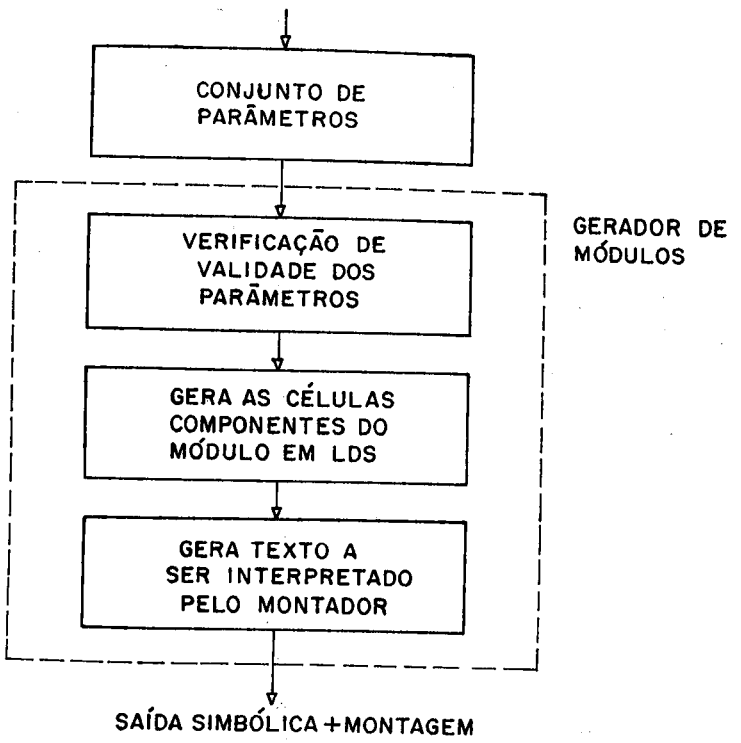


Fig. 5.8 - Geradores de Módulos do GPO

5.2 O Conjunto de parâmetros.

Como já foi comentado, o uso de parâmetros possibilita que o módulo seja utilizado em mais de um projeto, pois pode-se invocar o gerador para produzir os módulos dentro das características desejadas para um certo projeto. Em /DEM 87/ encontra-se uma classificação de

parâmetros, qual seja, módulos podem ter parâmetros estruturais ou de composição.

Parâmetros estruturais dizem respeito à própria forma do módulo, como o número de bits de saída de uma ROM, a organização de uma RAM, o número de bits de uma PO, etc.

Os parâmetros de composição referem-se aos atributos dos módulos. Assim, a informação de que um somador possui registrador de entrada é uma composição do somador; o número de operações de uma ULA, o uso de circuitos pseudo-nMOS ou dinâmicos, a conexão ao barramento feita por transistores n ou por portas de transmissão são também exemplos de parâmetros de composição.

Na classificação de parâmetros de /DEM 87/ falta um tipo fundamental, qual seja, o conjunto de parâmetros elétricos. Estes são os que determinam a dimensão de transistores, de linhas de alimentação, de buffers de relógio e de outras partes do circuito, de maneira a se atingir o compromisso desejado entre área, freqüência e consumo para uma dada estrutura.

Um gerador que ofereça os três tipos de parâmetros prima pela generalidade. Parâmetros estruturais e de composição permitem uma otimização arquitetural do circuito. Decide-se por exemplo a quantidade de paralelismo, o número de barramentos, o uso ou não da pré-carga, etc. Os parâmetros elétricos são o ajuste fino do módulo, pois uma vez decidida a estrutura , pode-se otimizar transistores importantes até atingir o desempenho desejado.

Existem geradores de módulos em que não existe a parametrização elétrica, pelo fato de que as células constituintes do módulo são projetadas tendo em vista uma freqüência máxima de utilização (/DEM 86/). Quando da troca de tecnologia não se tem informação precisa do comportamento elétrico do módulo.

Em /MAR 86/ somente os drivers de saída e as linhas de alimentação são parametrizáveis. Em /CAR 88b/ e /HES 89/ os parâmetros elétricos confundem-se com aqueles de composição: o usuário é obrigado a trocar de buffer quando quiser uma maior capacidade de corrente.

O gerador de partes operativas possui os três tipos de parâmetros. Em termos globais (parâmetros estruturais), pode-se parametrizar o número de barramentos, o número de fases de relógio e o número de bits. Extrapolando-se o conceito de parâmetro global, a tecnologia também é parametrizável.

Cada módulo possui diferentes parâmetros de composição, que variam de 8 para o somador a 4 para os registradores mais simples. Estes parâmetros de composição abrangem conexão (a que barramento estão conectadas a entrada e a saída), tipo de célula presente no módulo (registrador de entrada, saída indicadora de zero, deslocador de saída) e se a conexão ao barramento ou de realimentação (registradores) é feita com apenas um transistor n (barramentos a pré-carga) ou com um par n p (TG).

Eletricamente, cada módulo possui 2 ou mais transistores em um caminho crítico, seja ele a carga/descarga do barramento ou a linha de carry. Transistores destes caminnos podem ser parametrizados para se atingir a velocidade desejada de operação da PO.

O levantamento de parâmetros possíveis de serem oferecidos aos usuário do gerador foi tarefa difícil: era preciso descobrir, dentre as inúmeras opções de implementação, aquelas que eram passíveis de serem parametrizadas e aquelas que poderiam ter uma estrutura fixa. O conjunto de parâmetros disponíveis é analisado no capítulo 7, juntamente com a linguagem de entrada do gerador e uma discussão sobre o alcance e limitações da ferramenta.

Como observação final sobre parâmetros, cabe observar que o número de parâmetros de um módulo é fixo. Se forem necessários mais parâmetros, deve-se reescrever a rotina C que gera o módulo. Isto traz limitações que impedem que as variações de parâmetros tracem uma linha contínua no gráfico da figura 5.9. Por exemplo, os barramentos de dados dos módulos gerados são sempre em metal, correndo perpendicularmente às linhas de comando.

Esta aproximação para realização do gerador de partes operativas não é tão geral como seria se, ao invés de rotinas geradoras de células em uma estrutura parametrizável, houvesse um gerador de células livre, como em /LIN 87/, onde o conjunto de entradas, saídas ou desempenho elétrico da célula pode ser gerado com muito mais variações.

O metódo utilizado no GPO contudo tem a vantagem de ser rápido e possuir parâmetros suficientes para a gama de problemas que pretende solucionar. Além disto, é muito superior ao simples uso de uma biblioteca de células. Tome-se como exemplo 8 módulos do GPO capazes de implementar o algoritmo de MDC. Contando-se os parâmetros de composição, tem-se a possibilidade de gerar 34 células distintas, sem se levar em conta as combinações possíveis entre os parâmetros. Somando-se os parâmetros globais, o total sobe a mais de 200 células, não se incluindo a possibilidade de alteração de tamanhos de transistores, o que elevaria o número possível de células a mais de 500.

Note-se o elevado potencial das bibliotecas de funções utilizadas, visto que a manutenção de uma biblioteca de 500 células para apenas 8 módulos diferentes é impensável, o que não dizer quando da inclusão de mais módulos ao GPO.

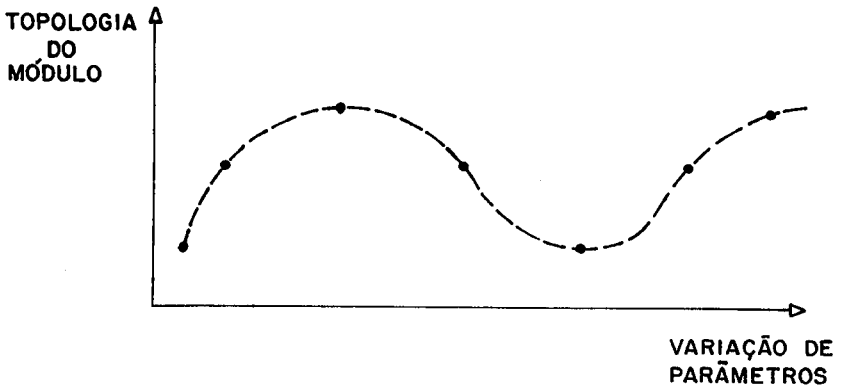


Fig. 5.9 - Parâmetros e topologia

Quando um gerador de células totalmente parametrizável estiver disponível no GME, basta trocar as rotinas geradoras de células atuais por chamadas ao gerador. Poder-se-ia aumentar o número de parâmetros do gerador, tornando a ferramenta ainda mais poderosa.

6 A AVALIAÇÃO ELETRICA

Fornecer uma realimentação ao usuário sobre o desempenho elétrico do layout a ser gerado é fundamental para ferramentas de implementação, visto que o objetivo é se ter em mãos projetos de alto desempenho e confiáveis. Ao mesmo tempo, a avaliação automática de desempenho representa um grande desafio, tanto na descoberta de modelos de circuitos quanto na manutenção da independência da tecnologia.

O processo tradicional de extração com posterior simulação elétrica é factível apenas para pequenos circuitos; um número elevado de transistores exige um tempo enorme de CPU, impedindo um projeto em que haja interação em tempo real do usuário com o sistema de CAD.

Nem todos os circuitos de uma PO são interessantes de serem simulados. Certamente a cadeia de carry é mais lenta que a carga de um registrador, estando ali o caminho crítico a ser otimizado. Sugere-se o uso não de uma extração indiscriminada, mas de um monitoramento acurado sobre partes realmente críticas do circuito. Estas são detectadas pelo próprio construtor da ferramenta, prevendo o comportamento elétrico do circuito.

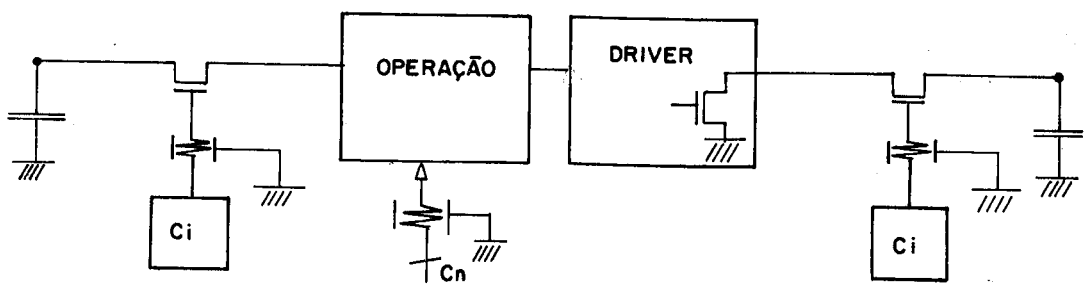


Fig. 6.1 - O Modelo da PO

Cabe definir o que é parte crítica em relação a

POs. Fundamentalmente, o circuito de uma parte operativa pode ser modelado como o da figura 6.1, onde existe um caminho de conexão do barramento a um dado módulo, um elemento de operação e um elemento de ligação ao barramento de saída.

Os comandos de conexão (S_i,C_i) devem carregar uma linha RC distribuida no caso de polisilício, ou apenas um capacitor se os comandos forem em metal. Por elemento de operação entende-se tanto células de cálculo de funções quanto células de armazenamento de dados. O que conta é a existência de uma rede de transistores e um caminho crítico.

Identificam-se 4 pontos interessantes para serem estudados sob o ponto de vista elétrico: o atraso de ativação dos comandos de conexão, o atraso de propagação da entrada até o amplificador, o tempo de carga ou descarga do barramento pelo amplificador e a potência consumida a uma dada freqüência de operação.

Localizados os pontos de interesse nos módulos da PO, resta definir como atuar em relação à avaliação elétrica. Uma primeira aproximação pode ser feita pelo uso de facilidades decorrentes dos módulos serem programas em linguagem C. Poder-se-ia ter em cada módulo uma rotina encarregada de gerar uma descrição SPICE ou ARAMOS do circuito que estiver sendo gerado, para posterior simulação elétrica.

Esta solução foi abandonada pelo excessivo tempo que o usuário deveria dispender para simulação de cada um dos módulos até a obtenção do caminho crítico e tentativa de otimização do mesmo.

Preferiu-se desenvolver modelos eletricamente equivalentes (dentro de uma certa tolerância) mas computacionalmente simples, para que a análise do comportamento elétrico fosse imediata, permitindo um uso

efetivamente interativo da ferramenta com o usuário.

6.1 Os modelos utilizados.

A busca de modelos e algoritmos para avaliação temporal (atrasos, tempo de subida, etc) rápida de um circuito não é recente, já existindo diversos programas que calculam atrasos entre redes de transistores, como em /OUS 85/, /PUT 82/ e /JOU 83/.

Estes trabalhos baseiam-se muito em /RUB 83/, onde um modelo para atrasos em redes RC ou de transistores é discutido. Um dos principais problemas do modelo de Rubinstein-Penfield é o de não tomar em consideração a não linearidade dos transistores MOS, substituindo-os por uma resistência linear equivalente a um determinado ponto de operação. A aplicação do método de Rubinstein-Penfield conduz a resultados com tolerâncias de 20% ou mais.

Uma solução para a não linearidade no comportamento dos transistores MOS como resistência encontra-se em /HOR 83/. Ali e em /OUS 85/ tenta-se também levar em consideração a influência da forma de onda de entrada em uma porta durante o chaveamento, o que até então não era considerado nos cálculos de atraso.

Em /SAK 83/ tem-se uma fórmula para cálculo de atrasos em linhas de conexão tipo RC, em que se garante boa precisão para uma certa faixa de valores, enquanto que em /SHO 88/ e /GLA 85/ encontram-se outros estudos e fórmulas para o problema do atraso em redes de transistores.

Aparentemente, o estudo de /HOR 83/ é o mais completo, pois leva em consideração não-linearidades e formas de onda de entrada para estimar o atraso de chaveamento. Contudo, as fórmulas estimam os limites máximos e mínimos dos atrasos, havendo uma variação de até 20% entre estes limites.

Baseando-se em estudos anteriores (/CAR 88b/),

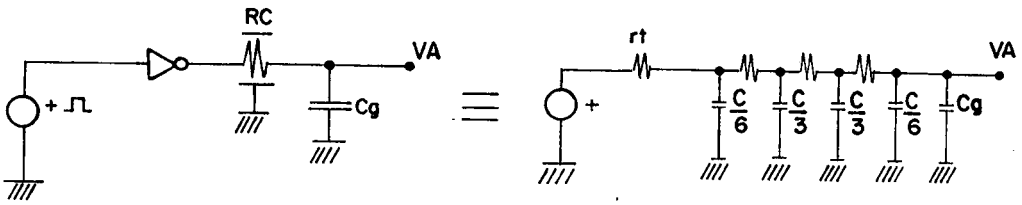
buscou-se uma aproximação ad hoc que permitisse uma precisão máxima, com erro menor do que 20%, sempre para uma classe específica de problemas. Apesar de altamente empírico, o método mostrou-se confiável e adaptado as exigências de precisão e parametrização quando da troca de tecnologia.

6.2 As linhas de comando.

Cada um dos três caminhos críticos da PO sofreu uma análise detalhada de comportamento, e a partir dai montou-se o esquema geral.

Se as linhas de comando forem em polisilício, estas sofrerão um atraso devido ao resistor-capacitor (RC) distribuído da linha. Sakurai (/SAK 83/) propõe que o circuito da figura 6.2a pode ser aproximado pelo da figura 6.2b, onde rt é a resistência equivalente do amplificador, Cg a capacitância de carga (provavelmente a porta do próximo estágio), R a resistência total da linha e C a capacitância total da linha. Para esté circuito, o delay ou tempo para a tensão no ponto A excursionar de 10 a 90% de seu valor final é dado por

$$t_{\varphi o} = 1.02*R*C + 2.21*(rt*C+R*Cg+rt*Cg).$$



(a) (b)

Fig. 6.2 - Modelo de Sakurai

Quando rt<<R, o driver pode ser aproximado por uma fonte de tensão, enquanto que se rt>>R, o driver comporta-se como uma fonte de corrente não linear. Por

trabalhos realizados com o gerador de ROMs e o gerador de PLAs constatou-se que a fórmula de Sakurai tem ótimos resultados (erro menor do que 3%) para a faixa de problemas em que rt<<R, ou seja, a carga/descarga do capacitor destino é dominada pela resistência da linha.

Um problema em relação à fórmula de Sakurai diz respeito à forma de onda de entrada, visto que todas as relações são deduzidas para resposta ao salto, situação que dificilmente se apresenta em CIs.

A condição rt<<R para que a fórmula seja aproveitável é devida ao fato de que, quando rt se aproxima de R ou é muito maior, o cálculo do tempo de atraso é dificultado (não se tem um valor limite para V_{cm}), e como a maior parcela da resposta é devida à rt, o erro aumenta. Como o transistor é um elemento não linear, o cálculo de sua resistência equivalente fica prejudicado.

Para linhas de comando em polisilício de POs maiores do que 4 bits, geralmente a condição rt<<R é plenamente satisfeita. O cálculo de rt é dado por

$$rt=L/(W*m*C_{ox}*(V_{gm}-V_{th})),$$

visto que V_{cm} é pequeno durante a maior parte do atraso. Após a aplicação do salto na entrada do transistor o capacitor mais à esquerda da linha atingirá rapidamente a tensão final, devido à baixa constante de tempo rt*C (comparado a outros estágios da linha). Portanto, a maior parte da carga ou descarga da linha será feita com V_{cm} próximo de seu valor mínimo (desprezível).

Resta ainda considerar o efeito da onda de entrada. Como, por hipótese, rt<<R, mesmo que o gate de rt tivesse em sua entrada uma onda lenta, a constante de tempo (RC) dominante seria a da linha. Além disto, supõe-se que o usuário deseja o comando o mais rápido possível, e portanto a excursão do sinal à entrada de rt será feita o mais aguda possível. Por simulações, variações de um salto para uma rampa de 3ns a entrada de rt causaram pouco efeito (<2%) à

forma de onda de resposta ao final da linha, o que confirma o acima exposto, na hipótese de ter-se $dv_{in}/dt \ll dv_{out}/dt$.

Assim, considera-se que esta aproximação para as linhas de comando é suficiente. A de Sakurai fornece o tempo de 10 a 90% (de subida ou descida) necessário para ligar o último bit da linha de comando de um certo módulo. É preciso levar em conta que à linha de comando são acrescidas capacitâncias de gate, tantas quanto forem necessárias para modelagem de um bit da PO.

O método da soma é a simples inclusão dos capacitores de gate de um bit à capacitância total da linha. Como a soma é repetida ao longo de todos os bits, os polos intermediários criados não são significativos, e podem ser imaginados como também eles distribuídos (figura 6.3).

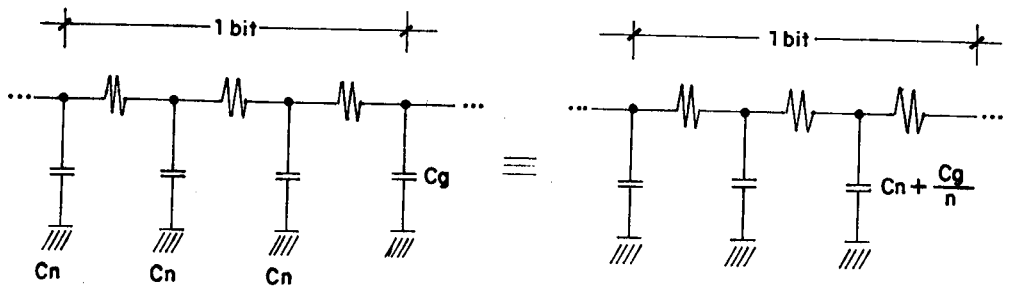


Fig. 6.3 - soma de capacitâncias na linha de comando

6.3 O atraso de propagação.

As dificuldades para um modelo elétrico eficiente surgem ao se considerar como se deve avaliar o atraso de carga de registradores, da cadeia de carry, do cálculo de Pi, Gi, etc. Neste caso, a fórmula de Sakurai é totalmente inadequada, enquanto que o método de Rubinstein-Penfield peca pela substituição do transistor por uma resistência linear.

Um circuito MOS como o da figura 6.4 pode ser aproximado por uma rede de transistores como a da figura 6.5. Imagina-se que ambas as chaves estejam ligadas, e portanto o caminho crítico é aquele da figura 6.5. O que interessa é o cálculo do atraso para a tensão de 5V em CB1 chegar à CB2.

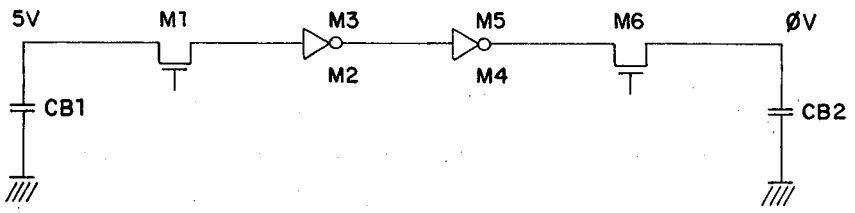


Fig. 6.4 - Circuito MOS

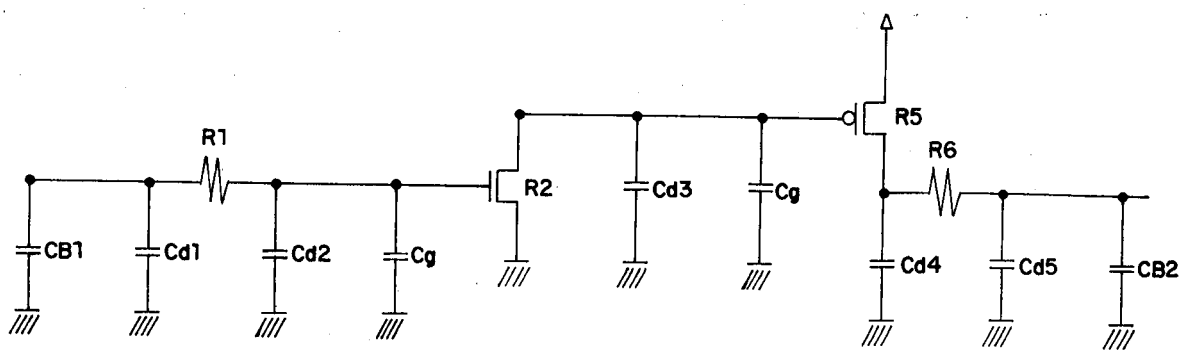


Fig. 6.5 - Circuito equivalente

O caminho crítico é uma sucessão de chaveamentos entre transistores p e n, até o último. Transistores em série com o caminho crítico são considerados estáveis (totalmente ligados ou totalmente desligados), colaborando apenas com as capacitâncias de dreno e fonte e a resistência quando ligados. Pode-se adicionar ao circuito as resistências e capacitâncias do polisilício de

interconexão, as resistências de contato, etc. Em conseqüência, um caminho crítico é definido como na figura 6.6.

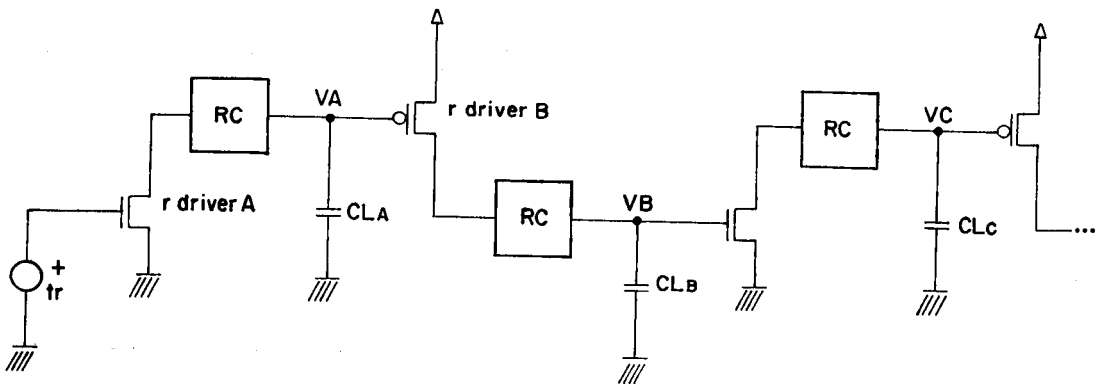


Fig. 6.6 - Caminho crítico

O bloco RC pode ser uma linha longa de polisilício, uma rede de transistores em série ou uma mistura de ambas. Para cada seção do caminho crítico define-se uma resistência de driver, uma rede RC, uma capacitância de carga, um atraso e um tempo de subida ou descida. Com estes elementos pode-se calcular o atraso de 50% e o de 90% da tensão final.

Para gates em que o ganho da rede n é aproximadamente igual ao da rede p ($W/L_n * m_n * C_{ox}$ = $W/L_p * m_p * C_{ox}$), o ponto de chaveamento encontra-se em 50% de Vin ou 2.5V. Para circuitos como os da figura 6.6, o delay total será dado pela soma dos delays individuais de cada seção (figura 6.7).

Obviamente, para circuitos em que os ganhos são distintos, os pontos de chaveamento também o são, provocando que a medida do delay deva ser feita em um ponto diferente. Se $W/L_n = W/L_p$, tem-se o ponto de chaveamento em 1.71V. Como regra geral, pode-se estimar o ponto de

chaveamento através da equação (/WES 85/)

$$Vinv = VDD+V_{tp}+V_{tn}*sqrt((m_n*W/L_n)/(m_p*W/L_p))/(1 + sqrt((m_n*W/L_n)/(m_p*W/L_p))).$$

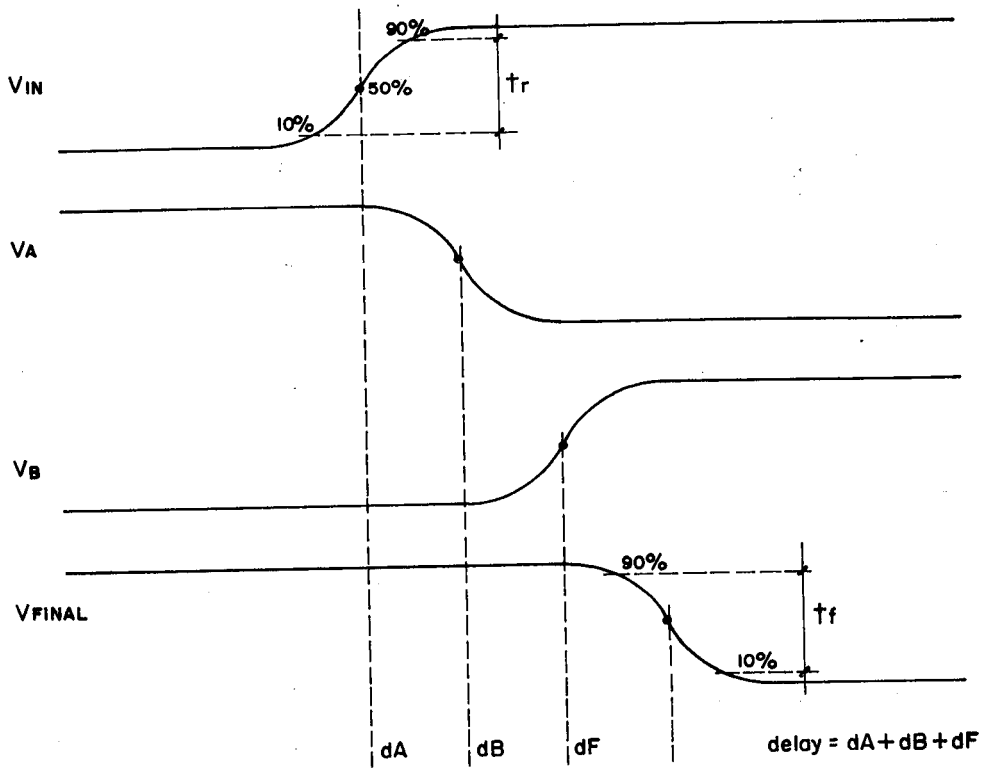


Fig. 6.7 - Definição de delay, tf e tr

O tempo de subida (tr) e o tempo de descida (tf) indicam um comportamento da onda de saída de cada seção. Se tr e tf da entrada do estágio são pequenos, tem-se praticamente um salto, caso contrário aproxima-se a forma de onda na entrada por uma rampa. Uma dificuldade para modelamento de dispositivos MOS é a dependência de cada seção da figura 6.6 com a curva de resposta da seção anterior. Note-se que a resistência efetiva de um transistor depende da tensão de porta e consequentemente da forma de onda na sua entrada. No caso de um salto, a

resistência efetiva é baixa durante quase toda a transição de saída. Se a entrada é lenta o transistor deverá fazer a (des)carga das capacitâncias do nó de saída enquanto está parcialmente ligado, significando uma resistência maior.

Sendo assim, o cálculo do atraso passa pelo cálculo das resistências efetivas, que por sua vez necessita do cálculo do tempo de subida ou descida da onda de entrada. Em relação à figura 6.6, pode-se dizer que a rede RC influencia no atraso de subida ou descida dos gates, e portanto influencia também no chaveamento.

Intuitivamente surge um algoritmo para cálculo do atraso, com a hipótese de que o circuito é uma rede sem laços de realimentação como na figura 6.2:

-com tr inicial, descobre-se a resistência efetiva do driver;

-com a resistência efetiva, a rede RC e CL_n descobre-se o tempo de descida de V_n;

-com o tempo de descida de V_n descobre-se a resistência efetiva do driver B, e prossegue-se assim até o último estágio.

Pelas características inerentes dos circuitos MOS de possuirem resistores e capacitores (significando diversos polos, com poucos zeros significativos), a resposta de uma rede MOS é uma soma de exponenciais. Embora a curva resultante exata seja difícil de se expressar em uma equação, pode-se dizer que existe uma resistência tal que $V(t) = V_0 e^{(-t/T)}$, $T = RC$, de maneira que a onda de saída real e a exponencial equivalente encontram-se em algum ponto. Fazendo-se este ponto o de 90% da tensão de saída, tem-se o tempo de subida da onda real, embora não se tenha a forma de onda exata desta.

Considerando-se que este erro de forma de onda não é significativo, procedeu-se com simulações do circuito da figura 6.8, variando-se a linha RC (para variação de tr e tf de Vi), o tamanho do inversor de saída e a

capacitância de carga CL. O campo de variação estudado foi aquele possivelmente encontrado em circuitos de POs típicas (a menos dos circuitos de amplificação dos barramentos), qual seja, CL<15*Cg (onde Cg é a capacitância de entrada de um inversor mínimo), W/L do driver até 10 vezes maior que o mínimo e R da mesma faixa da resistência do driver, o que ocasionaria o pior caso de estimativa de delay conforme visto anteriormente.

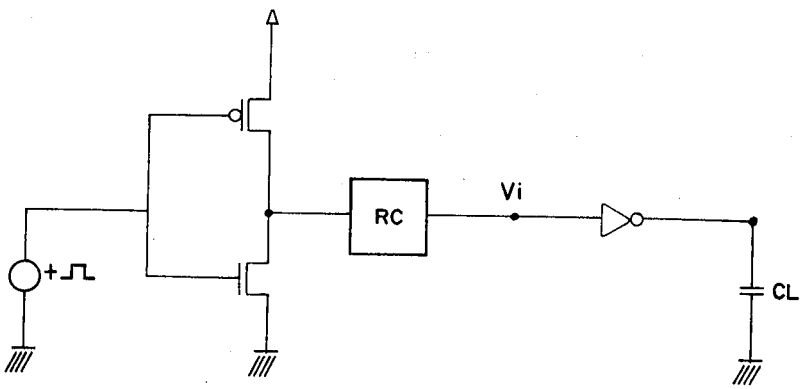


Fig. 6.8 - Circuito para levantamento da resistência efetiva

Com este conjunto de simulações buscou-se a resistência linear equivalente, ou seja, a resistência R tal que a exponencial cruzasse a função de saída a 90% da tensão final.

Como o chaveamento depende da tensão de entrada, do W/L do estágio e da capacitância de saída, dois dos parâmetros foram mantidos constantes enquanto se variava o terceiro, de modo a se identificar a contribuição de cada um individualmente.

Primeiro, com a variação de RC obtiveram-se diferentes tempos de subida e descida em Vi, caracterizando-se diferentes pontos. Depois provocou-se uma variação de CL, mantendo-se W/L constante. O resultado foram as curvas expostas no gráfico da figura 6.9, entre

resistência efetiva e tempo de subida.

Claramente, a resistência equivalente é uma reta dependente de CL. Quanto maior este último, menor a inclinação. Para um mesmo tr, a diferença de resistências equivalentes é muito maior na curva à 1Cg do que a 10Cg. A interpretação deste fenômeno é a seguinte: quanto maior CL, maior sua influência na curva de resposta, fazendo com que a carga ou descarga do capacitor seja feita em uma tensão de porta máxima durante a maior parte do tempo.

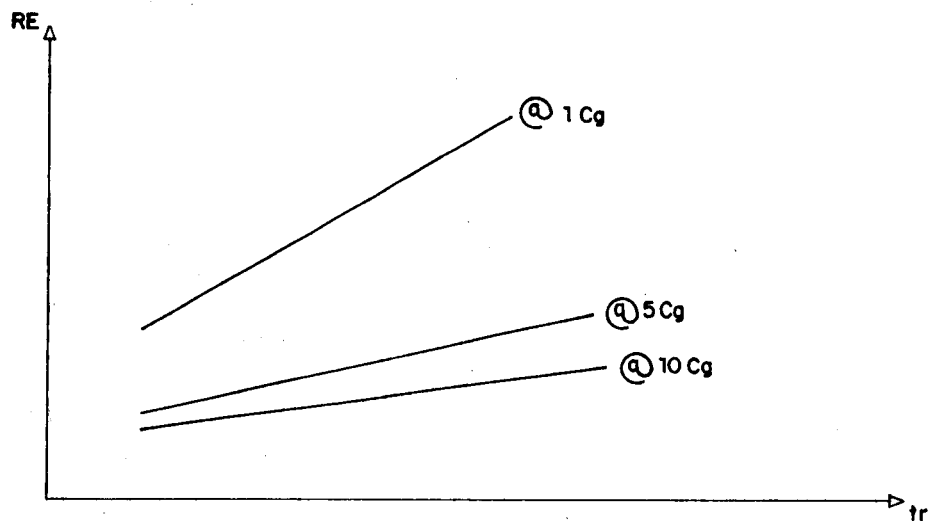


Fig. 6.9 - Gráfico de RE x tr em VI

Não foram levados em consideração tempos de subida ou descida enormes, como 100ns, nem CLs muito grandes, como 2pF. Para as células da PO, os tempos de subida e as capacitâncias de carga situam-se exatamente na faixa abrangida pelas simulações; a avaliação elétrica perde em generalidade mas ganha em eficiência.

A curva para CL = 5Cg encontra-se muito mais próxima da curva de 10Cg do que do meio do gráfico. As curvas de RE por tr são portanto influenciadas por CL de maneira diferenciada. Tem-se, para tr em nanosegundos, Cg em picofarads e RE em kilohms que

RE = 4.5164*tr + 10.3738 @ 1Cg;
RE = 1.0064*tr + 5.6521 @ 5Cg;
RE = 0.5676*tr + 5.0463 @ 10Cg.

Pode-se escrever RE = A*tr + B, onde A e B são funções de CL. Resolvido o problema da relação entre tr (tf) e CL, resta incluir o efeito de W/L na resistência efetiva do driver. W/L influencia na entrada (um W/L maior implica em um Cg maior, aumentando tr) e na saída (um W/L maior conduz mais corrente). O efeito na entrada já foi equacionado, pois basta considerar que Cg influencia tr.

Por hipótese, o efeito de CL também já foi computado, e portanto escreve-se

RF = K1*RE + K2,

onde K1 e K2 são funções de W/L. Com os dados adquiridos de simulações, as funções K1 e K2 foram encontradas, onde K1=f((W/L)$^{-1}$), K2=g(1/(W/L)). A precisão no cálculo de RF confrontando-se as simulações foi de 1%, o que indica que as curvas e funções encontradas são boas aproximações.

Tendo-se a disposição fórmulas para, a partir de um tempo de subida ou descida encontrar-se a resistência efetiva do driver, pode-se agora proceder ao cálculo de delays. Com a resistência do driver, aplica-se o método de Rubinstein-Penfield, e encontra-se uma constante de tempo para o subcircuito do driver, rede RC e Cg do próximo estágio. Através de uma exponencial simples calcula-se o tempo de subida ou descida do próximo estágio, e sucessivamente.

Um problema encontrado dizia respeito a redes RC formadas por transistores. Supondo-os ligados, qual a resistência efetiva de cada um a considerar (figura 6.10)? É preciso levar-se em conta que a capacitância de dreno parasita forma com o transistor uma constante de tempo significativa para a rede, ou seja, como as capacitâncias são da mesma ordem de grandeza, uma variação em V_a demorará a fazer-se sentir em V_b. Portanto, o transistor encontra-se

com um V_{ds} significativo, e a resistência efetiva é de avaliação mais difícil. Contudo, empiricamente pode-se dizer que

$$R = L/(W*m_n*C_{ox}*(V_{gs}-V_{tn}-V_{ds}/2))$$

leva a resultados razoáveis, apesar da simplicidade do modelo resistivo.

Outra dificuldade do método reside no fato de que a resistência efetiva para 90% da tensão de saída produz péssima precisão para 50% da tensão, visto que a saída é uma soma de exponenciais e não uma única. A solução adotada foi a de se descobrir uma nova resistência para 50% a partir da resistência efetiva final (RF) de 90%. Encontrou-se assim um polinômio do terceiro grau.

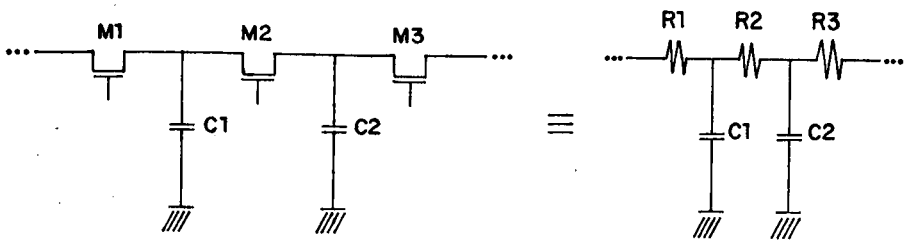


Fig. 6.10 - Rede RC de transistores

O método foi aplicado em uma série de circuitos típicos, entre os quais o da figura 6.11, que simula uma cadeia de carry de 4 bits. O erro encontrado para os circuitos foi sempre menor do que 3% na avaliação de tr e tf, e menor que 9% na avaliação do delay total. Este último percentual deve-se a baixa precisão do polinômio utilizado, que apresentava erros de 15% em algumas resistências. Apesar do alto índice de erro no cálculo do delay total, este ainda é aceitável tendo em vista a própria variação de parâmetros de processo.

Este método para cálculo do atraso tem a vantagem da simplicidade e precisão. Entre as desvantagens

é preciso citar a faixa limitada de aplicação e a necessidade de novas baterias de simulações a cada troca de tecnologia para levantamento dos coeficientes das equações.

Algumas topologias de circuitos não podem ser abordadas pelo método, como por exemplo um transistor n conectado à VDD ao invés de um transistor p. A tensão final não é mais VDD, mas (VDD-V_{tn})/lambda, e portanto novas resistências e equações devem ser obtidas por novas baterias de simulações. Por experiência local, a quantidade de simulações (28) e o levantamento de parâmetros leva em torno de 2 semanas, o que seria então o tempo necessário para obtenção de novas constantes para o modelo, permitindo a troca de tecnologia.

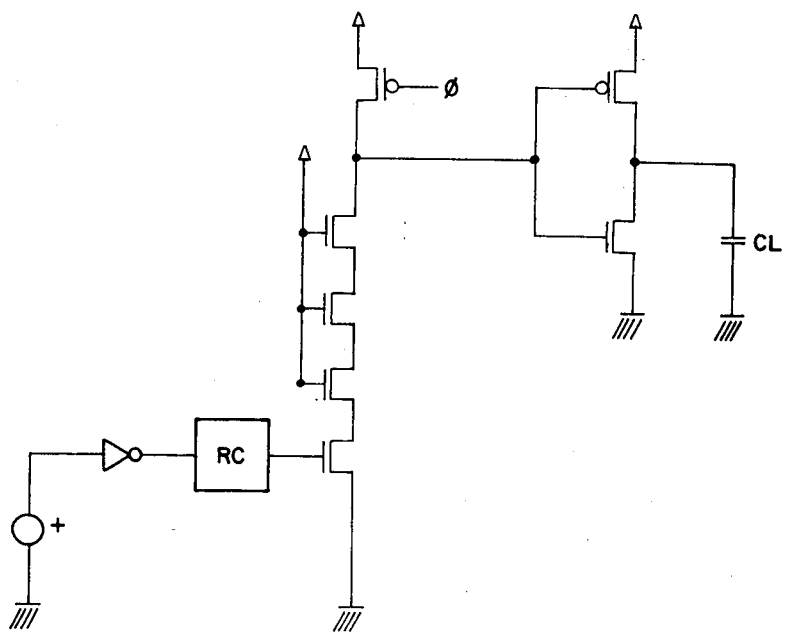


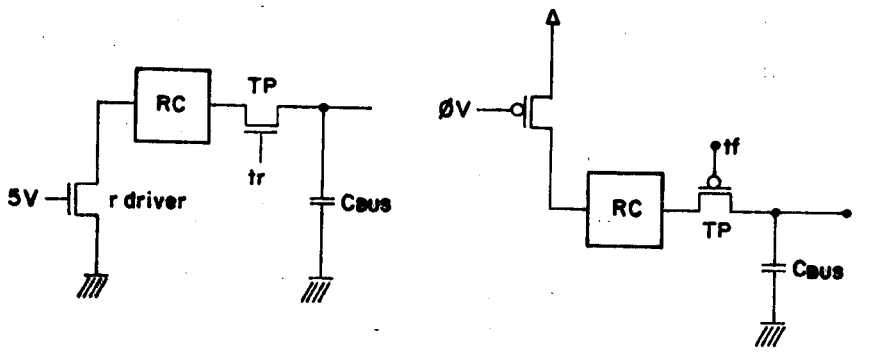
Fig. 6.11 - Circuito de carry

As simulações foram todas feitas considerando-se parâmetros médios. Poder-se-iam criar diferentes versões de uma mesma tecnologia utilizando-se parâmetros elétricos de melhor e pior caso do processo em questão, aumentando-se assim a precisão do avaliador.

6.4 A conexão ao barramento.

A conexão ao barramento é modelada pelos circuitos da figura 6.12, onde em a tem-se o circuito para barramentos a pré-carga, em b o circuito para barramentos simples, onde a carga/descarga é feita pelas próprias células isoladamente.

Fundamentalmente, os circuitos consistem em um transistor de driver, uma linha RC, um transistor de passagem (n, p ou ambos) e o capacitor do barramento. Supõe-se a entrada do driver estável, visto que a operação deve já ter sido concluída quando da ativação do comando para ligação do módulo ao barramento. A rede RC mais tipicamente é um conjunto de transistores em série, como nos deslocadores à esquerda e à direita do gerador de partes operativas (/CAR 89/). O transistor de passagem será ativado por uma linha de comando em polisilício, e portanto tr ou tf terão séria influência no seu modo de condução. O barramento é em metal, e assim pode-se desconsiderar sua resistência em comparação com a resistência série de rdriver + TP.



(a) (b)

Fig. 6.12 - Conexão ao barramento

Observou-se que para POs grandes, em que o capacitor de barramento era significativo (2pF), a fórmula

de Sakurai tinha boa precisão (erro menor do que 4.5%). Dificuldades advinham quando do uso de POs curtas, em que o cálculo das resistências equivalentes tinha maior influência no resultado final.

Decidiu-se pela aplicação do método descrito no ítem 6.3. Primeiro, com a altura da PO, calcula-se tr ou tf conforme ítem 6.2. Com este valor pode-se avaliar a resistência efetiva do transistor de passagem como foi anteriormente demonstrado. A resistência do driver e da rede de transistores é calculada pela suposição de que V_{ds} é significativo, ou seja, $C_{bus}>10*C_{drenos}$. Esta situação só não é encontrada em POs muito curtas (por exemplo, dois registradores lado a lado). Assim,

$$rdriver = L/(W*m_n*C_{ox}*(V_{gs}-V_{tn}-V_{ds}/2)),$$

sendo V_{ds} tomado como o início da região de saturação, ou seja, $V_{ds}=V_{gs}-V_{tn}$.

A partir daí basta a aplicação do método de Rubinstein-Penfield. O circuito da figura 6.13 apresenta a constante de tempo T dada por

$$T=rd*Cd1+(rd+R1)*Cd2+(rd+R1+R2)*(Cd3+C_{bus}).$$

Com esta constante, o tempo necessário para a tensão no barramento excursionar de 10 a 90% de seu valor final é dado pela exponencial simples, qual seja,

$$t = 2.1054*T.$$

As fórmulas de cálculo abrangem todos os tamanhos possíveis de PO. Para o modelo de carga/descarga do barramento, os testes até agora realizados indicam um erro máximo de 3%.

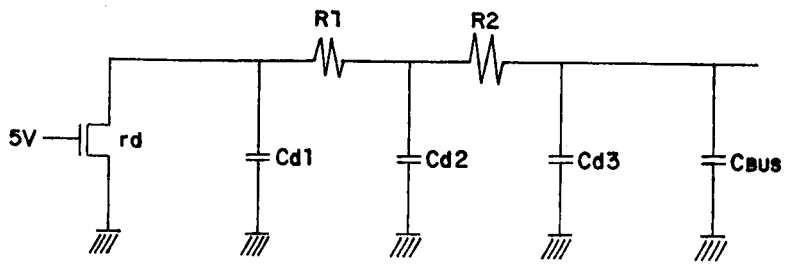


Fig. 6.13 - Circuito equivalente para conexão ao barramento

6.5 A potência.

Para a PO existem duas potências de interesse: aquela de carga/descarga dos barramentos e aquela presente dentro dos módulos. A primeira é facilmente calculada, bastando para isto ter-se a capacitância total do barramento e a freqüência de utilização. A potência será dada por (C_{bus} em pico-Farads)

$$P = VDD^2 * C_{bus} \quad microW/MHz.$$

A este valor deve ser agregada ainda a potência estática devido à corrente reversa (/WES 85/). Cada módulo porém pode ter uma implementação diferente. Dependendo se CMOS complementar, a pré-carga, pseudo nMOS ou dominó, as fórmulas para o cálculo da potência serão diferentes. É preciso então que cada módulo produza uma saída descrevendo que modelo de circuito utiliza para cálculo da potência.

Com os modelos de circuitos aqui apresentados foi desenvolvido o avaliador de desempenho de POs. O avaliador tem boa resposta em termos de velocidade e precisão, servindo aos propósitos anteriormente determinados. Apesar do uso do avaliador ser vantajoso, ele não impede o usuário de utilizar ferramentas tradicionais como um extrator e um simulador elétrico.

As fórmulas e precisão encontradas serão validadas por medidas nos circuitos implementados. Como o modelo é empírico, uma realimentação através de circuitos

físicos é fundamental. Possivelmente, por ter seus dados levantados do simulador elétrico, o mesmo tipo de discrepância existente entre o simulador e o circuito real deverá estar presente no avaliador. Contudo, a simples existência de um modelo que funcione motiva a pesquisa de métodos de avaliação mais precisos e menos limitados.

Os resultados de simulações, as tabelas para levantamento das resistências equivalentes e os circuitos de teste encontram-se em /CAR 89/. No próximo capítulo encontra-se o formato de entrada e saída do avaliador.

7 OPERAÇÃO DA FERRAMENTA

Como já foi anteriormente comentado, o número de parâmetros do gerador de partes operativas não é infinito, e portanto partes operativas diferentes são geradas como variações de uma estrutura de base. A topologia consiste em barramentos em metal, cortados perpendicularmente por comandos em polisilício. Estes tornam-se bastante inconvenientes quando usados em POs altas (24, 32 bits), visto que a linha RC dominará o atraso do comando por menor que seja a resistência do driver.

Existem então três soluções possíveis. A mais evidente consiste no uso de duas camadas de metal, a primeira para comandos e a segunda para alimentação, relógios e barramentos. Esta solução implica na troca da tecnologia utilizada pelo gerador, embora o editor simbólico suporte dois níveis de metal. As células dos módulos deveriam ser redesenhadas simbolicamente para aproveitamento das duas camadas, mas muito possivelmente haveria poucas alterações a fazer na parte de geração de módulos e no compilador de linguagem de entrada.

Outra solução seria o uso de barramentos em polisilício e não em metal, deixando esta camada para os comandos, como é feito em /DUC 86/. E muito mais fácil, devido à topologia do layout, colocar-se um buffer entre os módulos do que entre células de um mesmo módulo, pois por ali deve passar uma linha de comando. A desvantagem desta aproximação é o custo maior em área devido aos buffers, além do que para POs com muitos módulos (longas), o elevado número de buffers também estaria no caminho crítico do atraso.

A terceira opção seria a colocação de um amplificador na outra ponta do comando, como na figura 7.1 (/SHO 88/). O driver do comando tem de carregar/descarregar a ponta terminal da linha somente até um pequeno percentual da tensão final. O amplificador ajudaria o driver nesta

extremidade, tornando o comprimento da linha de comando, em termos do atraso que ele representa, menor do que o real. O problema reside no projeto do amplificador, que será um circuito muito sensível a ruídos e a variações de processos.

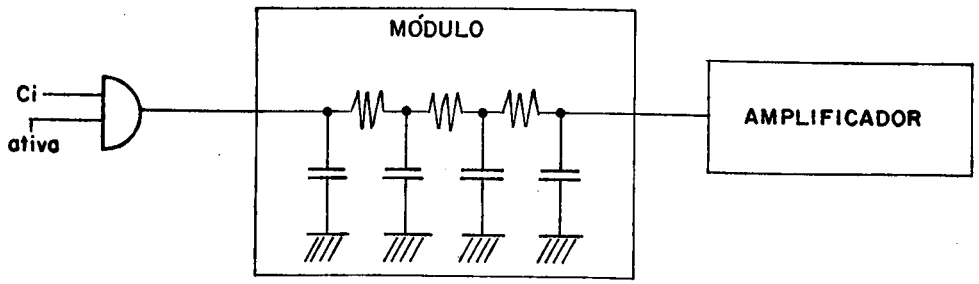


Fig. 7.1 - Comando e amplificador

De qualquer maneira, as três soluções são factíveis. Por ser a de mais fácil execução (no presente momento) e pela necessidade de compatibilidade entre diferentes MPCs, optou-se pela terceira, onde não é necessária a troca de topologia do gerador.

As principais limitações da ferramenta dizem respeito ao número de barramentos, que podem ser no máximo 7. Em princípio, devido ao exposto no capítulo 4, existem três barramentos de dados (A, B e C) e até quatro fases (Phi1, Phi2, Phi1N ou Phi3, Phi2N ou Phi4).

Alguns módulos utilizam obrigatoriamente Phi1, outros Phi1 e Phi2, sendo o uso de Phi1N e Phi2N condicional (parametrizável). Existem módulos que não utilizam nenhuma fase, e então poder-se-ia aproveitar as fases não utilizadas como barramentos de dados extras, embora 3 seja um número suficiente pelas considerações do capítulo 4.

Dentro da estrutura comentada até agora, a

liberdade é praticamente total, dependendo apenas do criador do módulo e dos parâmetros que ele deseja oferecer. Na figura 7.2 tem-se a linguagem de entrada do gerador. Entre os parâmetros globais encontram-se o número de bits, a tecnologia, os barramentos e as fases utilizadas. A seguir aparecem diversas listas de parâmetros elétricos, de composição e de conexão.

```
NAME : POTESTE;
TECHNO : gren;
BUS : A,B;
Nbits :8;
PHASE: Phi1;
/* este é um comentário */
COMP:1> TG=1;
COMP:2> TG=0,INV_ENT=1,REG_ENT=1, SHIFT_SAI=1,ZERO_OUT=1,POSITIVO=1;
ELECT:2> Psp=2.5, Psn=2.5;
ELECT:4> Pkkn=1.0, Pkkp=2.76, GND=3.0;
CONNEC:3 > INTERVAL 2
           [0,3]{ E:A
                  S:A|B
                }
         [4,7]{ E:A
                S:B
              };
ELEMENTS {
        OPERATOR(somest,*,COMP:2,ELECT:4)
        REGISTER(rAestsimp,*,*,*)
        OPERATOR(entESQasc,CONNEC:3,COMP:2,*)
        OPERATOR(entassinc,CONNEC:3,COMP:2,*)
        OPERATOR(amplibus,CONNEC:3,*,*)
        REGISTER(rdinasp,*,*,*)
        OPERATOR(entassinc,CONNEC:3,COMP:1,*)
        REGISTER(rsemiestat,CONNEC:3,COMP:1,*)
        REGISTER(rAestsimp,*,*,*)
        OPERATOR(precarga,*,*,*)
        USR(celula,*,*,*)
        };
```

Fig. 7.2 - Linguagem de entrada

As listas são depois referenciadas pelo módulo para personalização. No caso do usuário ainda não estar familiarizado com a ferramenta, ou se ele ainda não possuir uma estimativa do que parametrizar, os parâmetros default podem ser invocados através de um asterisco (*).

A medida que cada novo elemento vai sendo

encontrado ele é gerado. Portanto, a posição final dos módulos na PO resultante é idêntica a da descrição.

As listas de conexão servem tipicamente para circuitos de entrada e saída da PO. Por exemplo, de uma entrada de 16 bits, 8 devem ir para um registrador conectado ao barramento A pela sua parte baixa, enquanto que 8 devem ir para outro registrador conectado ao barramento B pela sua parte alta.

Existe a possibilidade de inclusão de um ou mais elementos tipo USR, indicando que ali deve ser posicionado um módulo não presente entre as rotinas do gerador, mas sim fornecido pelo usuário.

Se o usuário não especificar nenhum parâmetro global, o barramento de dados utilizado será o A. Se em algum módulo existe a necessidade do uso de, por exemplo, Phi1, e esta não foi especificada como um parâmetro ativo, uma mensagem de erro será enviada e o programa abortará. O compilador e as rotinas de módulo foram construídos tendo em vista uma validação da PO. Apesar das células serem projetadas, extraídas, simuladas e terem sidos submetidas ao DRC antes de sua inclusão no gerador, o uso de parâmetros pode trazer situações conflitantes, que são verificadas exatamente pelo compilador ou pelo gerador de módulos. A liberdade fornecida ao usuário implica nesta necessidade de validação.

O compilador e o gerador preocupam-se exclusivamente com a parte operativa; o algoritmo que ela deve executar está fora do alcançe destas ferramentas, sendo necessário um simulador de máquinas de estados finitos (/FRI 89/) ou um simulador como o da linguagem KAPA (/WAG 87/). Para isto, bastaria incluir nas rotinas geradoras de módulos um trecho onde seria gerado o texto com o circuito equivalente do módulo para as referidas linguagens, de maneira a se poder trabalhar em fase posterior com os respectivos simuladores. Na figura 7.3

apresenta-se uma rotina do compilador indicando a lista de mensagens de erro fornecidas por diferentes módulos.

```
char *erros_msg(err)    /* função que retorna um ponteiro para */
int err;                /* a mensagem de erro.                 */
{
static char *nomerros[]={
       "erro inexistente.",
       "erro estrutural: módulo amplibus necessita Phi1",
       "erro estrutural: módulo rsemiestat necessita Phi1",
       "erro estrutural: módulo rdinesp necessita Phi1 e Phi2",
       "erro composição: módulo rdinesp com TG necessita Phi1N e Phi2N",
       "erro estrutural: módulo precarga necessita Phi1",
       "erro composição: módulo rAestsimp não aceita mais de um intervalo",
       "erro composição: módulo rBestsimp não aceita mais de um intervalo",
       "erro composição: módulo rCestsimp não aceita mais de um intervalo",
       "advertência: módulo entESQasc deve ser o mais à esquerda da PO",
       "erro composição: módulo somest não aceita mais de um intervalo",
       "erro estrutural: módulo somest necessita Nbits múltiplo de 4",
       "erro composição: somest necessita REG_ENT: só existe 1 barramento",
       "advertência: com 2 barramentos é recomendável REG_ENT ou mais fases"


/* colocam-se aqui mais mensagens de erro. Não esqueça de colocar a
   última vírgula */
       };
return((err<1||err>13)? nomerros[0]:nomerros[err]);
}
```

Fig. 7.3 - Lista de erros

A saída do GPO é um conjunto de células simbólicas e um texto de montagem, que tem o nome da PO seguido do sufixo .mod (figura 7.4). Tem-se a opção então de chamar o montador simbólico, o gerador de máscaras e o avaliador elétrico. A célula englobante é sempre chamada de PO, sendo acrescida de .mod para o montador, .cel para o gerador de máscaras e .avl para o avaliador.

O compilador foi desenvolvido com o uso dos programas lex e yacc do ambiente EDIX V (UNIX like). Apesar de se poder fazer a linguagem de entrada como macros, a opção pela construção do compilador é vantajosa pois não existe a necessidade de recompilação para cada PO a ser gerada, e além disto pode-se facilmente alterar a linguagem de entrada. Os textos lex e yacc do compilador encontram-se

nos anexos 6 e 7 respectivamente.

```
V E1_0 = ( C1_0[][] ) # 4;
V E1_gbl = ( E1_0[][])#1;
V Mfnt2_0 = ( Cfnt2_0[][] # Cfnt2_1[][] #
              Cfnt2_2[][] # Cfnt2_3[][]) #1;
V E2_0 = ( C2_0[][] ) # 4;
V E2_fnt = ( E2_0[][])#1;
H E2_gbl = ( Mfnt2_0[][] + E2_fnt[][] ) +1;
V E3_0 = ( C3_0[][] ) # 4;
V E3_gbl = ( E3_0[][])#1;
V E4_0 = ( C4_0[][] ) # 4;
V E4_gbl = ( E4_0[][])#1;
V E5_0 = ( C5_0[][] ) # 4;
V E5_gbl = ( E5_0[][])#1;
V E6_0 = ( C6_0[][] ) # 4;
V E6_gbl = ( E6_0[][])#1;
V E7_0 = ( C7_0[][] ) #4;
V E7_1 = ( C7_1[][] ) #4;
V E7_2 = ( C7_2[][] ) #4;
V E7_3 = ( C7_3[][] ) #4;
V Efnt7_0 = ( C7_4[][] # C7_5[][] #
              C7_6[][] # C7_7[][] )#1;
V E7_4 = ( Efnt7_0[][]) #1;
V E7_5 = ( C7_8[][] # C7_9[][] ) #2;
V E7_6 = ( C7_10[][] ) #4;
V Efnt7_1 = ( C7_11[][] ) #3;
V E7_7 = ( Efnt7_1[][] # C7_12[][] ) #1;
V E7_8 = ( C7_13[][] ) #4;
H E7_gbl=(E7_0[][]+E7_1[][]+E7_2[][]+E7_3[][]+
          E7_4[][]+E7_5[][]+E7_6[][]+E7_7[][]+
          E7_8[][])+1;
V E8_0 = ( C8_0[][] ) # 4;
V E8_gbl = ( E8_0[][])#1;
H PO = ( E1_gbl[][] + E2_gbl[][] + E3_gbl[][] +
         E4_gbl[][] + E5_gbl[][] + E6_gbl[][] +
         E7_gbl[][] + E8_gbl[][] )+1;
```

Fig. 7.4 - Texto do montador

A introdução de novos módulos é tarefa bastante simples. O administrador desenha uma célula simbólica, indicando os parâmetros elétricos e de composição que irá necessitar (/MAR 89/). Feito isto, a célula deve passar pelo extrator para garantia de que o layout simbólico foi introduzido corretamente.

Com o programa ggcel, o administrador obtém uma

rotina C capaz de gerar a célula. O compilador fornece então duas rotinas, busca_I(parâmetro) e busca_F(parâmetro), que devolvem os valores de parâmetros globais e de composição ou elétricos respectivamente. O administrador deve então escrever uma rotina C que verifica a validade dos parâmetros globais da PO (por exemplo, se o módulo exige Phi1 e ela não está presente), chama as rotinas de células utilizadas no módulo e gera um texto para o montador no arquivo de saída. A verificação de parâmetros e o nome do arquivo de saída são fornecidos por rotinas do compilador.

Feito isto, basta a inclusão do nome do módulo na lista de módulos, compilação das rotinas C (célula e módulo) e ligação com o compilador como um todo. Com um pouco de treino, módulos simples como registradores podem ser incluídos em menos de dois dias, entre o desenho do registrador e seu uso no GPO.

O compilador foi escrito de tal modo a que novos parâmetros possam ser facilmente encaixados: quando do surgimento de um módulo com, por exemplo, o parâmetro INVERSOR-DE-SAIDA, basta colocar o nome do parâmetro na lista de parâmetros a que ele pertencer (elétricos ou de composição). Após recompilação, o gerador de partes operativas pode ser utilizado e o parâmetro instanciado.

Maiores detalhes do programa e de seu uso podem ser obtidos em /CAR 89/. No anexo 8 encontra-se o manual de utilização e o manual do administrador do GPO. É importante observar que o método para gerar módulos seguido no GPO pode ser utilizado para gerar qualquer tipo de módulo com parâmetros. Por exemplo, poder-se-ia substituir bibliotecas de células tradicionais por bibliotecas de funções. Tudo o que se tem a fazer é desenhar as células simbolicamente (para usar o ggcel) e escrever uma descrição yacc para o compilador da linguagem de entrada do módulo. A biblioteca de funções estaria parametrizada e pronta para uso.

7.1 Os módulos existentes.

A primeira preocupação foi chegar-se em módulos capazes de implementar os circuitos dos exemplos estudados no capítulo 4. Assim, os módulos pesquisados foram:

-registrador estático simples;
-registrador semi-estático;
-registrador dinâmico especial;
-banco de registradores;
-constantes;
-registrador-deslocador;
-registrador-contador;
-registrador com saída para a PC;
-registrador com saída de zero;
-entrada assíncrona;
-saída assíncrona;
-amplificador de barramento;
-pré-carga;
-somador-subtrator estático look-ahead;
-somador-subtrator manchester;
-somador-subtrator manchester look-ahead;
-gerador de funções para ULA;
-chaves de barramento;
-conexão do barramento com a PC.

Cada módulo possui diferentes parâmetros. Por exemplo, o somador-subtrator estático tem a opção do registrador de entrada, inversor de entrada para subtração, deslocador de saída, sinal de saída zero e sinal de saída positiva. Dos módulos estudados, até o momento da conclusão deste trabalho estavam operacionais e integrados ao gerador o somador-subtrator estático com carry look-ahead, os três tipos de registradores, a entrada assíncrona, o amplificador de barramento e a pré-carga.

Apesar de poucos, estes módulos praticamente abrangem todos os pontos críticos para geração, como geração de células fantasmas, geração de células de conexão

(polisilício) diretamente em C, sem passar pelo editor, etc. Todos os módulos acima listados porém foram projetados até o nível de transistores, indicando-se também a parametrização possível (/CAR 89/). A inclusão destes módulos ao GPO é tarefa que demanda apenas um pouco mais de tempo.

A entrada e saída da PO não precisam ser feitas necessariamente pelos extremos do barramento, e nem sincronamente com relação a temporização dos mesmos. Existem os módulos entassinc e saiassinc que, através de linhas de polisilício, transmitem informações para cima ou para baixo, perpendicularmente ao barramento. Este tipo de ligação é útil não apenas para a parte de controle, mas também para o anel de pads (/TRU 86/).

A testabilidade da PO segue o esquema tradicional do scan-path (/MAR 86/, /MER 84/, /WIL 82/). Dois registradores deslocadores foram projetados. O primeiro é totalmente estático, mas exige um número maior de comandos para funcionamento. O segundo é menor e com menos linhas de comando, mas utiliza as duas fases não coincidentes e não superpostas para funcionamento.

Com os módulos acima listados pode-se implementar os 4 exemplos estudados e outros, como por exemplo a 2901, o algoritmo de Booth para multiplicação, etc. Com a inclusão de outros módulos (começando por um barrel-shifter) ou melhora dos já existentes (somador mais rápido), pode-se fazer até a PO do 68020, 386 e outros processadores com qualquer grau de complexidade. Tem-se uma ferramenta aberta e de largo espectro de atuação.

Observe-se que, quando é necessário um aumento de paralelismo com uma arquitetura pipeline, basta o uso das chaves de barramento e alguns registradores a mais para implementação de uma parte operativa pipeline. Pode-se ainda gerar duas POs em separado trabalhando sob a mesma máquina de controle, enfim, a criatividade do usuário é o

limite.

7.2 O avaliador elétrico.

Cada módulo do GPO quando instanciado gera também um texto para o arquivo PO.avl, indicando o número de transistores, as capacitâncias parasitas (Cg no comando e Cd no barramento) e uma série de dados para as equações do avaliador.

```
TECNOLOGIA = gren;
NBITS = 8;
Cgphi1 = 30.0;
Cgphi2 = 60.0;
VDD = 2.0;
GND = 5.0;
CdBusA = 600.0;
CdBusB = 180.0;
EQUACOES :
       MODULO registrador (1) : [
              Ccomando = 3.0;
              Ntransistores = 10;
              CAMINHO CRITICO n >2 INTERVALOS
                     {
                     TR(1)=0;
                     Dwl(1)=D;
                     Cg(1)=3.0;
                     Dwl(2)=3.0;
                     Cg(2)=5.0;
                     MSG($ o registrador tem somente dois caminhos$);
                     };
              CAMINHO BARRAMENTO A : n
                     {
                     MSG($  este é um comentario$);
                     TR=D;
                     Dwl=4.0;
                     TP=D;
                     Cg=CBUS(A);
                     };
              ];
       MODULO somador (2) : [
              Ccomando = 5.50;
              Ntransistores = 40;
              CAMINHO CRITICO n >4 INTERVALOS
                     { ...
```

Fig. 7.5 - Entrada para o avaliador

Cada módulo pode especificar um caminho crítico e

até três caminhos de barramento, um para cada um dos existentes na PO. Desta maneira, pode-se obter o atraso do caminho crítico, o atraso do barramento, a capacitância de entrada das fases para dimensionamento dos drivers de relógio, a potência consumida, etc. Na figura 7.5 tem-se o arquivo de entrada para o avaliador, e na figura 7.6 o arquivo de resposta.

```
GPO: MODULO DE AVALIAÇÃO ELETRICA
        Versão 1.0

Area da parte operativa: X= 1000, Y= 2013 micra.
Resultados globais:
Capacitância da fase Phi1: 0.21198 pF
Capacitância da fase Phi2: 0.41838 pF
Largura de VDD: 6  micra
Largura de GND: 15  micra
Capacitância do Bus A: 1.66254  pF
Potência do Bus A: 41.5635  microW/MHz
Capacitância do Bus B: 0.50334  pF
Potência do Bus B: 12.5835  microW/MHz

Analise do módulo registrador, elemento 1 da descrição.
dados da linha de comando:
        R= 24.156, C= 0.179614, delay = 5.45225
número de transistores acumulados: 10
$ o registrador tem somente dois caminhos$
$  este é um comentario$
Atraso de descarga do barramento A: 24.7862 ns
atraso combinacional do caminho crítico: 1.35684 ns

Analise do módulo somador,elemento 2 da descrição.
dados da linha de comando:
        R= 24.156, C= 0.317214, delay = 9.32915
número de transistores acumulados: 50
$ o somador tem quatro intervalos no caminho $
$  Analise do barramento A $
Atraso de descarga do barramento A: 27.5099 ns
$  Analise do barramento B $
Atraso de descarga do barramento B: 10.086 ns
atraso combinacional do caminho crítico: 4.42147 ns

Número total de transistores: 400
```

Fig. 7.6 - Saída do avaliador

O usuário, após gerar uma PO, pode alterar a entrada do GPO ou atuar diretamente sobre o arquivo PO.avl,

e chamando novamente o avaliador trabalhar iterativamente até obtenção da resposta elétrica desejada. Uma vez completo o estudo, retorna-se à linguagem de entrada do GPO e alteram-se os transistores conforme o anteriormente previsto. Economiza-se o tempo de geração de máscaras.

Também no avaliador pode-se ter comandos default, simbolizados pela letra D. Todas as grandezas (capacitores, tamanho de transistores) são expressas em fatores, ou seja, em múltiplos do tamanho mínimo. Isto foi feito para que o próprio valor do parâmetro elétrico do módulo servisse como entrada ao avaliador. No caso do usuário desejar alterar o arquivo de entrada, o único cuidado a tomar é não especificar uma capacitância de carga menor do que o menor transistor possível na tecnologia. Devido ao método empregado na avaliação elétrica (capítulo 6), cargas muito pequenas (.001pF) provocam o surgimento de tempos negativos no cálculo do atraso de propagação entre estágios. Para a tecnologia de Grenoble, por exemplo, o menor gate (tipo n) possui Cg = 0.0069pF.

O avaliador lê o arquivo <tecnologia>.avl para carregar os valores dependentes desta para o cálculo. Atualmente, 45 valores de tecnologia são utilizados (reporte-se ao anexo 9 para maiores detalhes). As listagens lex e yacc do avaliador encontram-se nos anexos 10 e 11.

7.3 Medidas de qualidade.

É importante avaliar-se o desempenho do gerador em relação à velocidade de geração e qualidade do layout gerado.

O gerador em si é bastante rápido; o gargalo do sistema encontra-se na expansão das diversas células simbólicas. Na máquina onde o sistema está instalado (ED 680, UNIX V, processador 68000 com 1.5 Mb de memória principal) a expansão de uma célula com 200 elementos (entre contatos, fios e transistores) leva aproximadamente

2 minutos. A mesma célula em máquinas "IBM PC-AT like" tomou apenas 8 segundos de CPU. Portanto, ao migrar-se o sistema para os PCs-AT espera-se um ganho de velocidade na ordem de 12 vezes. A geração de um circuito como o da figura 7.2 que hoje consome aproximadamente 10 minutos levaria menos de 1 minuto.

Para o circuito da figura 7.2 foram gerados 1054 transistores em 2.14 mm² (X = 1455 micra, Y = 1476 micra), o que significa aproximadamente 5 transistores em cada quadrado de 100 micra de lado, na tecnologia do MPC de Grenoble, canal 2 micra. Esta figura de mérito pode ser melhorada com o uso de um compactador para as células simbólicas. Considerando-se que na mesma área seria possível a inclusão de 6 inversores (28 por 50 na mesma tecnologia), observa-se que existe espaço para a otimização topológica das células antes da necessidade da compactação. Um projeto mais cuidadoso das células componentes dos módulos ajudaria muito.

Em termos de descrição, menos de uma página serviu para definir e implementar 1054 transistores, capazes de realizar a PO de um algoritmo. A PO pode ainda ser ajustada até obtenção do compromisso ótimo entre área, velocidade e potência pela mesma descrição de entrada, o que demonstra a versatilidade da ferramenta.

8 CONCLUSÃO

Neste trabalho apresentou-se uma ferramenta que deve colaborar no projeto automático de circuitos integrados VLSI. A ferramenta enquadra-se em uma metodologia de trabalho, procurando tirar o máximo proveito de sua especialização inerente.

O gerador de partes operativas é aberto, o que facilita sua evolução. Apesar do número de módulos atualmente disponíveis ser pequeno, a inclusão de novos elementos é extremamente simples. Cada novo circuito algoritmicamente complexo projetado no CPGCC poderá provocar o surgimento de mais módulos passíveis de serem agrupados.

Infelizmente, o gerador não é perfeito, existindo ainda uma série de limitações. A primeira diz respeito ao editor simbólico, que ainda carece de um compactador eficiente. Depois, nos cálculos realizados pelo avaliador elétrico, falta a inclusão de redes RC e da potência interna aos módulos. Esta limitação contudo é facilmente contornável, visto que os modelos desenvolvidos no capítulo 6 já os levam em conta. Basta portanto a inclusão de uma descrição de RCs no compilador de entrada da linguagem do avaliador.

Em termos de interface com o usuário, a entrada textual como exposta no capítulo 7 exige do projetista o conhecimento dos nomes dos parâmetros de cada módulo. Uma interface gráfica com menus certamente auxiliaria na escolha dos membros componentes de uma PO, sem que o usuário estivesse com o manual permanentemente a seu lado. Aparentemente esta não é uma limitação severa. Já que a ferramenta é específica, supõe-se que o usuário tenha um mínimo de treino antes de utilizá-la. Além disto, a linguagem textual de entrada é útil para documentação e pode ser gerada por uma outra ferramenta de nível superior. Portanto, mesmo com entrada via menus, a linguagem textual

deve ser mantida.

A maior limitação do gerador é o fato de que até o momento não se tenha implementado um circuito físico com ele, chegando-se apenas até as máscaras. Sem este passo o trabalho jamais poderá considerar-se concluído. E pelo uso do GPO que se comprovará o exposto neste trabalho, fazendo-se uma validação da ferramenta como um todo.

Por possuir independência da tecnologia, possibilidade de parâmetros e avaliador elétrico embutido, espera-se que a vida útil do gerador seja longa (6 anos?), e portanto haverá tempo para novos trabalhos e aperfeiçoamentos.

Em termos da ferramenta, existe espaço para implementação de um compactador simbólico como já mencionado. Além disto, pode-se melhorar certos pontos defeituosos do avaliador elétrico (atraso de 50% por exemplo), ou estudar-se mais a fundo outros modelos menos empíricos para não haver necessidade da bateria de simulações quando da troca de tecnologia. A inclusão de novos módulos é obviamente necessária e bem vinda.

Em termos de ambiente, é fundamental integrar o gerador na estação de trabalho SILEX em desenvolvimento pelo GME. Como a saída final é em linguagem RS, este trabalho, a menos da interface gráfica, deverá ser de fácil execução.

A ferramenta já possibilita a construção automática de metade de um sistema digital. A outra metade do sistema diz respeito à automatização da geração da parte de controle. Com ambas as ferramentas pode-se ter um Gerador de Máquinas Digitais Programável, tornando o caminho do processo de concepção ainda mais curto.

O gerador de POs, tendo uma linguagem de entrada definida, pode tornar-se alvo de ferramentas interpretadoras do nível algorítmico. Assim, a partir do

algoritmo é possível a extração da PO e posterior implementação. Como o gerador possui o avaliador, o laço de realimentação entre as ferramentas está garantido, e aquela de nível superior pode fazer sucessivas chamadas ao gerador até encontrar uma PO satisfatória.

Durante a vida futura do GPO, a figura do administrador é de extrema importância. Sob sua responsabilidade encontra-se a troca de tecnologia, a inclusão de novos módulos, a implementação de rotinas C para validação dos parâmetros daqueles e descrição do modelo elétrico equivalente para o avaliador.

Finalmente, o importante deste trabalho foi o estudo de uma metodologia de concepção e dos diferentes modelos de construção de partes operativas. Mesmo que as técnicas de implementação ou os módulos utilizados sejam brevemente abandonados, o ambiente em que foram desenvolvidos estará válido. As idéias contidas na ferramenta tem um tempo de vida maior do que o de seus próprios módulos.

ANEXO 1

DESCRIÇÃO DO ELEVADOR

```
SISTEMA elevador;
main()
{
        EXPORTAÇÃO Reset,emergência;
        IMPORTAÇÃO ACK_emergência[ACK0,ACK1];
        ENTRADA Topo,Térreo,emer_BOT;

/* procedimentos de inicialização   */

        while(!Térreo)
                Reset=1;
        Reset=0;

/* procedimentos para operação normal, dois processos e uma
PO base que trabalham concorrentemente */

        while(TRUE)
            COBEGIN
                controla_carro();
                avalia_pedido();
                {
                 while(TRUE)
                    if(emer_BOT==1)
                        emergência=1;
                    while(ACK_emergência[*]!=1)  /* espera que */
                                ;    /*todos os processos parem  */
                    executa_procedimentos_emergência();
                }
            COEND;
}

/* Comentários: o vetor de emergência é uma memória apenas
de leitura para esta máquina principal. Na PO desta mesma
máquina, existem dois processos e uma parte operativa base,
encarregada da inicialização e da vigilância em relação às
emergências.

        O vetor de emergência é um vetor apenas para o
processo main: para todos os outros processos ele aparece
como uma simples variável. */

PROCESS avalia_pedido()
{
        EXPORTAÇÃO: mem_S[n],mem_D[n],ACK1;
                        /* o sinal de emergência é interno     */
                        /* define-se um elevador de n andares  */
```

```
        ENTRADA: botão_D[n],botão_S[n];
                        /* recebidos os botões de cada andar   */

        ENTRADA: botão_andar[n];
                         /* recebidos botões internos  */

        IMPORTAÇÃO: apaga_memória, end_apg_mem[M],
                    emergência, Reset;
                        /*  importa um sinal que ordena o
desligamento da memória. end_apg_mem é um número binário
que indica qual memória deve ser apagada */

        LOCAL i;

/*  Algoritmo: fica para todo o sempre varrendo os botões
internos e externos,verificando emergência e verificando
quem deve ser desligado */

ACK1=0;
while(TRUE)
        {if(Reset)
                {for(i=2;i==n;i++)
                        mem_S[i]=mem_D[i]=0;     /* zera pedidos */
                 mem_D[1]=1;     /* faz pedido para descer ao térreo *
                }
         for(i=1;i==n;i++)
               {if(apaga_memória==1)
                        mem_S[end_apg_mem]=mem_D[end_apg_mem]=0;
                                /* reseta memórias */
                 if(botão_andar[i]==1)     /* se há pedido interno */
                        otimiza_direção(mem_S,mem_D,i);
                 if(botão_D[i]==1)   /* se há pedido para descer ext.
                        otimiza_direção(mem_S,mem_D,i);
                 if(botão_S[i]==1)  /* se há pedido para subir, ext.
                        otimiza_direção(mem_S,mem_D,i);
               }
         if(emergência==1) /* se existe emergência, para o
                break;                          procedimento normal */
        }
ACK1=1;  /* indica para o processo emergência
                                que está parado      */

/* Comentários: o otimizador decide qual memória (de subida
ou descida) deve ser marcada. Além disto, decide se haverá
uma marca efetiva ou se deverá esperar o próximo ciclo para
marcar, supondo uma movimentação do elevador em alguma
direção.
```

A sequência de operações parece ser esta mesmo: primeiro limpando as memórias e depois procedendo-se ao resto. Isto permite que sempre o pedido atual esteja gravado, e que não

existam desgravações espúrias.

O método escolhido para comunicação entre este processo e o principal foi o seguinte: primeiro encerramos todas as tarefas, depois verificamos se um dado sinal está ligado. Portanto é um conceito de POOLING, e não de interrupção. */

```
PROCESS controla_carro()
{
        IMPORTAÇÃO: mem_S[n],mem_D[n], Reset, emergência;

        EXPORTAÇÃO: apaga_memória, end_apg_mem[M],ACK0;
                    /* M é o log base 2 de n */

        LOCAL:      i, direção, andar_atual;

        SAIDA:      sobe,desce;   /* sinais para controle do motor *

        ENTRADA:    marca_andar, Térreo;

ACK0=0;
while(TRUE)
{
        if(Reset)
                desce_terreo();
        COBEGIN
                {verifica_pedido();              /*              */
                 COBEGIN                         /* esta parte   */
                        atende_pedido(direção); /* é a operação */
                        verifica_pedido();       /* normal */
                 COEND
                }
                {if(emergência==1)
                       {
                        salva_operação_atual();
                        para_andar();
                        ACK0=1;
                       }
                }
        COEND

}

desce_térreo()
{
        sobe=0; desce=1; direção=0;
        while(!Térreo)
            {desce=0;
             direção=1;
            }
        andar_atual=1;
```

```
}

PROCESS verifica_pedido()
{
/* lê as memórias de subida e descida */
        for(i=1;i<=n;i++)
                {
                 if(mem_S[i]==1)
                        {
                         sobe=1;direção=1;
                         while(andar_atual!=i)
                                 if(marca_andar==1)
                                         {
                                         andar_atual++;
                                         marca_andar=0;
                                         }
                        }
                 para_andar();
                }
}

para_andar()
{
        LOCAL: j;
        SAIDA: porta_aberta; /* poderia ser  mais um
                                processo          */

        porta_aberta=1;
        for(j=0;j<=3600;j++)  /* loop para delay */
                porta_aberta=0;
}

atende_pedido(direção);
{
        end_apg_mem[M]=(i*2)+direção;
     /* desloca uma casa à esquerda e soma com direção */

/* poder-se-ia escrever
        end_apg_mem[0..M]=i%
        end_apg_mem[M+1]=direção;     */

        apaga_memória=1;
        /*  a variável apaga_memória poderá ser  lida  pelo
processo avalia_pedido */
}


/*  A comunicação deste processo com o controlador de
emergência é através de um procedimento equivalente à
INTERRUPÇÃO: é como se a cada estado do autômato se
realizasse um teste sobre o sinal de emergência. */
```

/* Considerações sobre o elevador/sistemas digitais:

-O processo avalia_pedido fica constantemente monitorando uma variável para saber se deve ou não apagar alguma memória de pedidos. Portanto, a comunicação exige a leitura de algume memória escrita por um outro processo. Quando avalia_pedido decide que deve apagar, o endereço da memória é fornecido pela PO de outro processo. Esta é a maneira deste (uma das possíveis) influir na PO de avalia_pedido.

-Se existe uma memória que registra pedidos internos e externos, algum processo deve ler esta memória. No caso, o processo avalia_pedido tem as memórias, enquanto que verifica_pedido é o encarregado de lê-las.

-Uma vez atendido o pedido, este deve ser removido. O processo atende_pedido seta uma variável indicando a avalia_pedido que este deve apagar uma requisição. O endereço desta encontra-se disponível na variável end_apg_mem[], que por sua vez pertence ao processo controla_carro, sendo este pai de atende_pedido. */

ANEXO 2

DESCRIÇÃO DO PCIR

A máquina de estados do PCIR pode ser considerada Mealy ou Moore. Se o RI está na PO, então a PO é uma máquina de Moore. A PC é que é uma máquina Mealy, cujas saídas (comandos) dependem do estado atual e das entradas (código de instrução ci ou RI, registrador de instruções). Se houvesse um estágio para decodificação de instruções, então a parte de controle também seria Moore.

Por motivos de privilégio do PC e do ST (contador de programa e status register), estes foram retirados do banco de registradores, sendo criados registradores fantasmas de modo a se conseguir descrever o pipeline. A seguir a descrição completa do PCIR.

```
SYSTEM PCIR;

#define fetch    RI=M [PC];    \
                 PC=PCnext;    \
                 PCnext++;

#define imediato  COBEGIN      \
                  TE=M [PC];   \
                  PC=PCnext;   \
                  PCnext++;    \
                  COEND

#define rd       R[RI[4..7]]
#define rf       R[RI[0..3]]
#define op       RI[10]RI[9]RI[8]
#define co       RI[11..14]
#define I        ST[2]
#define carry    ST[0]
#define LB       ST[1]
#define Z        ST[3]
#define O        ST[4]
#define im       RI[15]
#define rk       K[RI[0..3]]
#define SP       R[13[*]]
main()
{

/* declarações */
Regmem  R[14][16];      /* 14 registradores de 16 bits */
Reg     RI[16],PC[16],ST[16]
```

```
Reg     PCnext[16],TE[16]; /*para facilitar interrupção de
                           acesso à memória */
Input   Reset,Interrupt; /*sinais de entrada de 1 bit cada*/
Const   K[3]={0,1,-1} /* ROM de constantes dentro da PO */

/* início do programa em si */
BEGIN
LOOP(1)
{
   COBEGIN
        PC=0;
        PCnext=0;
        ST=0;
   COEND
   LOOP(2)
      {
        COBEGIN          /* faz o primeiro fetch */
           RI=M[PC];     /* M[] é um operador que
           PCnext++;     /*acessa a memória*/

        COEND
        COBEGIN
           PC=PCnext;
           PCnext++;
        COEND
        LOOP(3)
           {
            CASE(Reset,Interrupt,ci,I,im) of
                0 X XXXX X X: break(LOOP(2)); /* Reset */
                1 0 XXXX X X: COBEGIN     /* interrupção */
                                M[SP]=PC;
                                PC=8H;
                                PCnext=8H;
                                SP=SP-1;
                              COEND
                              break(LOOP(3));
                1 1 0000 X 0: COBEGIN    /* ULA sem imediato */
                                calcula(co,rd,rd,rf);
                                fetch;  /* busca a próxima */
                              COEND      /* instrução */
                1 1 0000 X 1: imediato;
                              COBEGIN
                                calcula(co,rd,rd,TE[*]);
                                fetch;
                              COEND
                1 1 0001 X 0: COBEGIN /* ULA com constantes */
                                calcula(co,rd,rd,rk);
                                fetch;
                              COEND
                1 1 0010 X 0: COBEGIN  /* UIE */
                                opera_byte(co,rd,rf,LB);
                                fetch;
```

```
                        COEND
1 1 0010 X 1:           COBEGIN
                          imediato
                        COEND
                        COBEGIN
                          opera_byte(co,rd,TE,LB);
                          fetch;
                        COEND
1 1 0011 X 0:           COBEGIN  /* SEQ */
                          compara(rd,rd,rf,=);
                          fetch;
                        COEND
1 1 0011 X 1:           COBEGIN
                          imediato
                        COEND
                        COBEGIN
                          compara(rd,rd,TE,=);
                          fetch;
                        COEND
1 1 0100 X 0:           COBEGIN   /* SGT */
                          compara(rd,rd,rf,>);
                          fetch;
                        COEND
1 1 0100 X 1:           COBEGIN
                          imediato
                        COEND
                        COBEGIN
                          compara(rd,rd,rf,>);
                        COEND
1 1 0101 X 0:           COBEGIN   /* SGE */
                          compara(rd,rd,rf,>=);
                          fetch;
                        COEND
1 1 0101 X 1:           COBEGIN
                          imediato
                        COEND
                        COBEGIN
                          compara(rd,rd,TE,>=);
                        COEND
1 1 0110 X 0:           COBEGIN     /* BF */
                          calcula(+,PCnext,rf,1);
                          RI=M[PC];
                          PC=rf;
                        COEND
```

/* PCnext=RF+1. JUMP com valor em registrador. Como a próxima instrução é sempre executada, ela deve ser um NOP ou um invariante em relação ao JUMP. */

```
1 1 0110 X 1:           COBEGIN
                          TE=M[PC];
                          PC=PCnext;
                          PCnext++;
```

```
                                COEND
                                COBEGIN
                                  RI=M[PC];
                                  calcula(+,PCnext,TE,1);
                                  PC=TE;
                                COEND
/* veja que o JUMP sem imediato não quebra o pipe ! */
                 1 1 0111 X 0: COBEGIN  /* URC */
                                  rotaciona(rd,rf,carry,co);
                                  fetch;   /* acima tem-se o move ! */
                                COEND
                 1 1 0111 X 1: COBEGIN
                                  imediato
                                COEND
                                COBEGIN
                                  rotaciona(rd,TE,carry,co)
                                  fetch;
                                COEND
                 1 1 1001 X 0: M[rd]=rf;   /* store */
                                COBEGIN
                                  fetch;
                                COEND      /* 2 ciclos ! */
                 1 1 1001 X 1: COBEGIN
                                  imediato;
                                COEND
                                M[TE]=rf;
                                COBEGIN
                                  fetch;
                                COEND
                 1 1 1011 X 0: COBEGIN   /* push */
                                  M[rd]=rf;
                                  calcula(-,rd,rd,1);
                                COEND
                                COBEGIN;
                                  fetch;
                                COEND
                 1 1 1011 X 1: COBEGIN    /* push com imediato */
                                  imediato;  /* 4 ciclos */
                                COEND        /chamada de subrotina */
                                COBEGIN
                                  M[rd]=rf;
                                  calcula(-,rd,rd,1);
                                  PC=TE;
                                  PCnext=TE;
                                COEND
                                COBEGIN
                                  RI=M[PC];
                                  PCnext++;
                                COEND
                 1 1 1101 X 0: rd=M[rf];     /* load */
                                COBEGIN
                                  fetch;
```

```
                                COEND
                    1 1 1101 X 1: TE=M[rf];
                                  rd=M[TE];
                                  COBEGIN
                                    fetch;
                                  COEND
                    1 1 1111 X 0: calcula(+,rf,rf,1);    /* pop */
                                  rd=M[rf];
                                  COBEGIN
                                    fetch;
                                  COEND
                    1 1 1111 X 1: calcula(+,rf,rf,1);  /* pop=return *
                                  COBEGIN                /* 4 ciclos */
                                    PC=M[rf];
                                    PCnext=M[rf];
                                  COEND
                                  COBEGIN
                                    RI=M[PC];
                                    PCnext++;
                                  COEND
                                  COBEGIN
                                    PC=PCnext;
                                    PCnext++;
                                  COEND
                    DEFAULT     : COBEGIN   /* Instrução inválida */
                                    fetch   /* idêntico a um NOP */
                                  COEND
            }     /* fim do LOOP(3) */
      }           /* fim do LOOP(2) */
}                 /* fim do LOOP(1) */
}                 /* fim da descrição */

/* Algumas considerações a mais:
        -Existem 4 operadores complexos programáveis:
                calcula(),
                opera_byte(),
                compara() e
                rotaciona().
```

Estes poderiam ser novamente decompostos ou ser implementados diretamente como funções de uma PO base, como foi feito no PCIR real.

-M[x] é um operador complexo, pois aparece à esquerda e à direita do sinal de atribuição. Para tal, muito possivelmente ele não será uma PO base, mas precisará de nova decomposição. Veja que como M[] acessa a memória externa, ele é o encarregado de gerar os sinais de read e write.

-As operações triviais permitidas foram atribuição e incremento (= e ++). Todas as outras foram descritas como funções.

-Compara e calcula podem se unir em uma nova função computa(), visto que ambas podem utilizar a mesma ULA, por jamais estarem presentes em um mesmo ciclo de operação. */

ANEXO 3

MAPA DE PARTES OPERATIVAS

O mapa reflete o estudo feito sobre diferentes maneiras de se implementar um mesmo circuito. Quatro pontos principais foram analisados, quais sejam:

-a topologia, onde se decide o número de barramentos, a necessidade ou não de registradores à entrada da ULA e o uso ou não de pré-carga;

-a sincronização, isto é, o número de fases de relógio presentes para uma operação base da PO;

-a operação base, isto é, a operação que a PO é capaz de executar em um estado;

-o tipo de registrador.

Dependendo da temporização presente, pode-se ter registradores mais econômicos em termos de número de transistores e por extensão em área. Não foi feita uma análise de operadores, mas estes também dependem da temporização para otimização. Se a PO for dominada por operadores ou reg_ops, deve-se utilizar a temporização que facilite a otimização destes, para depois escolher o tipo de registrador. No caso em que os registradores dominam, procede-se ao contrário. Além disto, compromissos de velocidade em termos de transição de estados também estarão presentes (vide capítulo 4), e a decisão final é sempre um problema de se compatibilizar os diversos compromissos.

É importante observar que todo o mapa foi feito pensando-se nas operações x=x op y em paralelo com z=op(z). Às vezes, ocorrem casos em que para se completar estas operações passam-se por três operações base. Estas, contudo, não podem ser interrompidas na sua seqüência, pois fazem parte de uma única operação da máquina de controle! Sendo assim, do ponto de vista de sistemas digitais, estas operações são únicas, correspondendo sua execução a apenas um estado do sistema.

É possível que o relógio da PO seja então mais rápido do que o do controle, tomando importância o circuito

de interface (capítulo 4). Isto é equivalente a dizer-se que, a partir de um relógio principal, as fatias de tempo existentes na PO são mais numerosas que aquelas na PC.

O mapa não é definitivo nem completo, mas serve de base para tipos de circuitos que tornam-se alvo para o gpo. Praticamente todas as variações de implementação encontradas no mapa podem ser implementadas pelo uso do gpo na sua primeira versão. A medida em que novas temporizações são estudadas, pode-se atualizar o mapa e por extensão o gerador.

Para todos os registradores analisados utilizou-se um critério bastante conservador em termos de funcionamento, isto é, proibiu-se que a realimentação estivesse fechada quando da escrita ou leitura do registrador.

Na escrita, se o barramento é grande, o capacitor dominante será uma fonte de tensão que suplantará o inversor de realimentação, garantindo a escrita do nível correto. Na leitura, se o barramento não é muito capacitivo, a realimentação pode estar fechada e ainda assim o nível lógico correto será mantido.

Como não se sabe a priori o tamanho do barramento presente na entrada e na saída das células de registrador, e como as hipóteses acima são mutuamente conflitantes, proibiu-se a realimentação durante qualquer atividade de leitura ou escrita.

Todos os registradores foram então projetados tendo-se em mente garantia de escrita ou leitura para todos os tamanhos possíveis de barramento. Além disto, colocou-se onde possível um inversor de saída, de maneira que parametrizando-se o tamanho deste elemento permite-se alterar a velocidade de carga ou descarga do barramento.

As células dos bancos de registradores são tradicionais, encontradas em diversos projetos ou até mesmo em livros-texto. A célula analógica do banco de registradores merece um pouco mais de atenção. Esta é utilizada no PCIR (/TOD 86/), onde se permite a leitura para dois barramentos, com o valor e seu complemento em cada um. Como C1 e C2 podem não estar ativos ao mesmo tempo, deve-se garantir que o driver da célula consiga

descarregar o barramento antes que o inversor de realimentação troque de estado. Portanto, o divisor resistivo formado pelo transistor de passagem e o driver deve produzir uma tensão menor do que a tensão de threshold do transistor n para correto funcionamento. Isto provoca que o tamanho do driver seja de 4 a 5 vezes maior do que o do transistor de passagem.

Para escrita na célula, os barramentos são carregados com o valor da escrita e seu complemento, sendo posteriormente C1 e C2 ativados ao mesmo tempo. Este é portanto um banco de registradores com dupla leitura, mas simples escrita.

1)Topologia: 3 barramentos, sem pré-carga, sem registrador na entrada da ULA.

Sincronização: 2 fases não coincidentes e não superpostas. Um estado do sistema digital corresponde a um ciclo de Phi1, Phi2. Durante Phi1 estão ativos os comandos de seleção (S), enquanto que durante Phi2 estão ativos os de carga em registradores (C). A seleção de operação da ULA pode estar ativa todo este período.

Operação base: É possível x=x op y em paralelo com z=op(z).

Registradores: dinâmico especial, visto que é preciso manter os dados estáveis na saída da ULA.

2)Topologia: 3 barramentos, sem pré-carga, com registrador na entrada da ULA.

Sincronização: 2 fases não coincidentes e não superpostas. Um estado do sitema digital corresponde a um ciclo de Phi1, Phi2. Durante Phi1 estão ativos os comandos de seleção (S), enquanto que durante Phi2 estão ativos os de carga em registradores (C). A seleção de operação da ULA pode estar ativa todo este período.

Operação base: É possível x=x op y em paralelo com z=op(z).

Registradores: estático simples, estático especial, dinâmico simples e dinâmico especial.

3)Topologia: 3 barramentos, sem pré-carga, com registrador na entrada da ULA.

Sincronização: 3 fases. Em Phi1 acontece a seleção, em Phi2 a realimentação nos registradores para memorização e em Phi3 acontece a carga nos registradores.

Operação base: x=x op y em paralelo com z=op(z).

Registradores: Os semi-estáticos ocupam menor área e tiram proveito da temporização. Os outros contudo podem também ser utilizados, mas sem vantagens aparentes.

4)Topologia: 3 barramentos, com pré-carga, com registrador na entrada da ULA.

Sincronização: 4 fases. Em Phi1 é feita a pré-carga nos barramentos de entrada da ULA; em Phi2 é feita a leitura dos registradores, e também se poderia fazer a pré-carga da ULA, no caso desta ser dinâmica; durante Phi3 a ULA trabalha, ao mesmo tempo que se pré-carrega o barramento de saída da ULA; finalmente, em Phi4 escreve-se no registrador destino.

Operação base: x=x op y em paralelo com z=op(z).

Registradores: Qualquer dos estudados. Pode-se aproveitar a pré-carga e transformar a chave de entrada e saída dos registradores em apenas um transistor de passagem n, visto que só é necessária a descarga do barramento. No caso do uso de registradores semi-estáticos, as fases de memorização coincidiriam com as fases de pré-carga.

5)Topologia: 3 barramentos. com pré-carga, com registrador na entrada da ULA.

Sincronização: 2 fases, Phi2 e Phi4 fases falsas. A operação é idêntica à PO número 4, utilizando-se Phi2 e Phi4 para pré-carga. Os comandos só poderiam estar ativos quando Phi1=1 & Phi2=0 ou Phi3=1 & Phi4=0.

Operação base: x=x op y em paralelo com z=op(z).

Registradores: O preferido seria o dinâmico simples ou o estático especial. Como a pré-carga agora é curta, não se recomenda o uso de registradores semi-estáticos.

6)Topologia: 3 barramentos, com pré-carga, com registrador na entrada da ULA.

Sincronização: Duas fases não coincidentes e não superpostas; em Phi1 é realizada a pré-carga, enquanto que em Phi2 ativam-se ou os comandos de seleção ou aqueles de carga. Em relação a POs vistas até aqui, leva-se o dobro de estados para executar a mesma operação base.

Operação base: Reg1_ULA=x em paralelo com Reg2_ULA=y ou x=out_ULA.

Registradores: Semi-estático, com a realimentação ativa durante a pré-carga.

7)Topologia: 2 barramentos, sem pré-carga; registrador obrigatório na entrada da ULA.

Sincronização: 2 fases não coincidentes e não superpostas. Em Phi1 ativa-se a seleção dos registradores e a carga dos registradores da ULA; em Phi2 ativa-se a carga dos registradores destino. Veja que a operação é equivalente àquela de 3 barramentos! A diferença é que, eventualmente, os registradores devem estar conectados tanto ao barramento A quanto ao B. Portanto, 3 barramentos são úteis somente quando não existe um registrador na entrada da ULA. Para POs com mais de 16 bits, é certamente mais barato a inclusão do registrador do que a colocação de mais um barramento.

Operação base: x=x op y em paralelo com z=op(z).

Registradores: Estático simples, estático especial, dinâmico simples, dinâmico especial.

8)Topologia: 2 barramentos, com pré-carga, registrador obrigatório à entrada da ULA

Sincronização: Idênticas às POs 4, 5 e 6 acima.

Operação base: Idênticas às POs 4, 5 e 6 acima.

Registradores: Idênticas às POs 4, 5 e 6 acima.

9)Topologia: 1 barramento, sem pré-carga, registradores na entrada da ULA obrigatórios.

Sincronização: 3 fases não coincidentes e não superpostas. Em Phi1 ativa-se a seleção de um registrador e a carga de um registrador da ULA; em Phi2 faz-se o mesmo com outro registrador e com outra entrada da ULA, e em Phi3 escreve-se no registrador destino. Note-se que a ULA tem somente Phi3 para calcular sua resposta e colocá-la no barramento.

Operação base: x=x op y;

Registradores: Estático simples, estático especial, dinâmico simples onde a realimentação é comandada por Phi3 ou dinâmico especial, onde uma das barreiras temporais é comandada pelo OU de Phi1 e Phi2.

10)Topologia: 1 barramento, sem pré-carga, registradores na entrada da ULA obrigatórios.

Sincronização: 2 fases não coincidentes e não superpostas. Em Phi1 ativa-se a seleção de algum registrador, em Phi2 ativa-se a carga. Embora se consiga uma atribuição em um ciclo, para uma operação completa de adição são necessários 3.

Operação base: Reg1_ULA=x ou x=y ou x=out_ULA.

Registradores: Os semi-estáticos são vantajosos pela economia de área.

11)Topologia: 1 barramento, com pré-carga. Registrador na entrada da ULA obrigatório.

Sincronização: 2 fases não coincidentes e não superpostas. Em Phi1 ocorre a pré-carga, em Phi2 ativa-se ou um comando de seleção (S) ou um comando de carga (C). Uma simples atribuição toma agora dois ciclos de Phi1-Phi2. Uma adição levaria 6 ciclos para completar-se.

Operação base: Barramento=x ou x=barramento.

Registradores: Novamente os semi-estáticos são vantajosos, e pode-se colocar na sua comunicação com o barramento apenas um transistor n, devido à pré-carga.

12)Topologia: 2 barramentos, bloco de registradores, pré-carga obrigatória.

Sincronização: 4 fases. Em Phi1 e Phi3 faz-se a pré-carga; em Phi2 lê-se o conteúdo dos registradores e em Phi4 escreve-se no banco de registradores.

Operação base: x=x op y.

Registradores: Banco de registradores estáticos, célula analógica, ou banco de registradores semi-estáticos. Neste caso, a realimentação é permitida em Phi1 e Phi3. Se o barramento for complementado, pode-se, ao custo de duplicação dos barramentos, utilizar o banco totalmente estático.

13)Topologia: 2 barramentos, banco de registradores, pré-carga obrigatória.

Sincronização: 2 fases não coincidentes e não superpostas. Uma soma leva 3 ciclos de Phi1-Phi2.

Operação base: Reg1_ULA=x em paralelo com Reg2_ULA=y.

Registradores: banco de registradores semi-estáticos.

14)Topologia: 1 barramento, bloco de registradores, pré-carga obrigatória.

Sincronização: 2 fases não coincidentes e não superpostas. Em Phi1 tem-se a pré-carga, em Phi2 a leitura ou escrita no banco.

Operação base: Reg1_ULA=x.

Registradores: banco de registradores semi-estáticos.

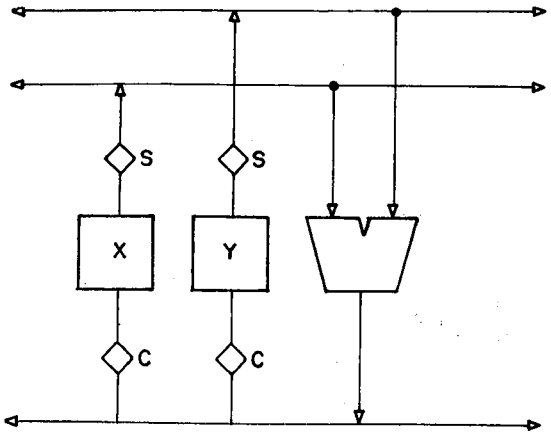
15)Topologia: 1 barramento complementado, banco de registradores, pré-carga obrigatória.

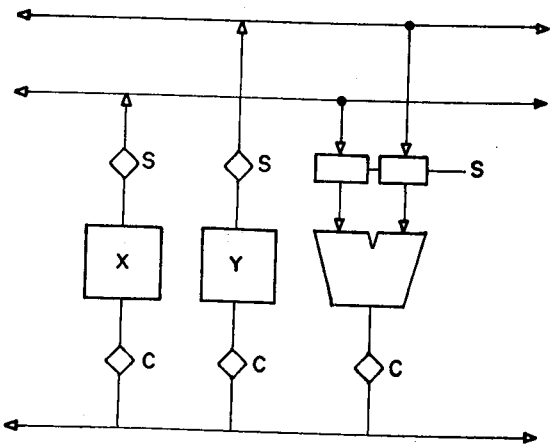
Sincronização: 4 fases. Em Phi1 faz-se a pré-carga, em Phi2 ativa-se o comando de leitura, em Phi3 ativa-se o sense-amplifier e em Phi4

ativa-se a carga no destino. Uma tranferência leva um ciclo de 4 fases, enquanto que uma adição leva 3 ciclos de 4 fases. Se houvesse mais um barramento complementado, então a adição seria possível em 2 ciclos de 4 fases.

Operação base: Reg1_ULA=x. Uma operação como x=y deve passar pela ULA, tomando então dois ciclos de 4 fases.

Registradores: banco de registradores estáticos.





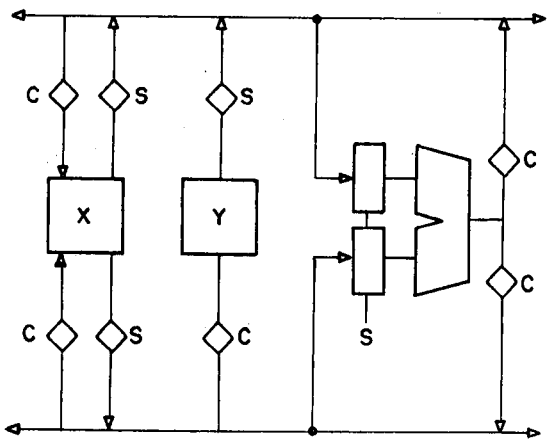


Fig. A3.1 - Topologias: a)3 barramentos, ULA puramente combinacional; b)3 barramentos, ULA com registrador de entrada; c)2 barramentos, ULA com registrador de entrada.

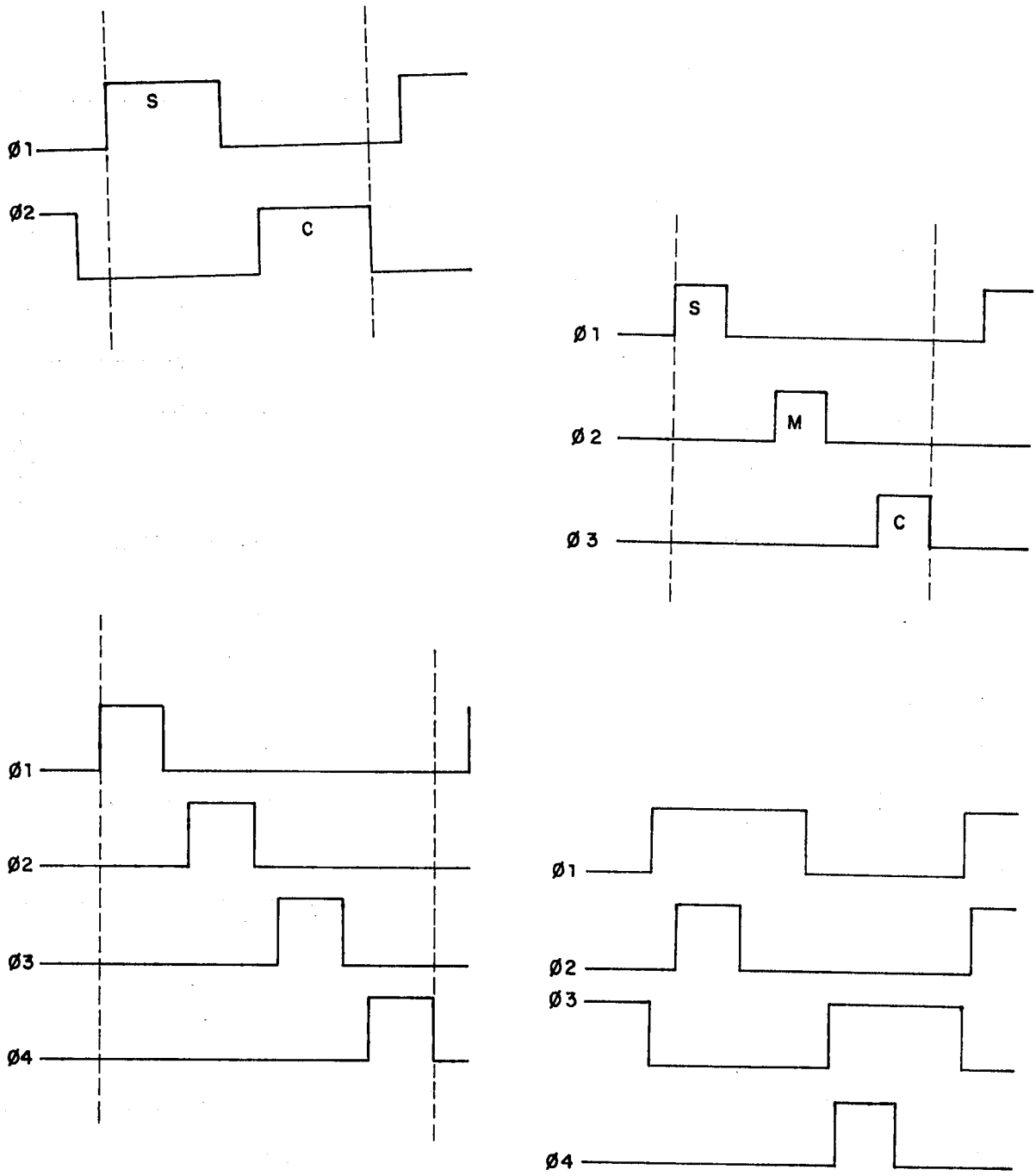
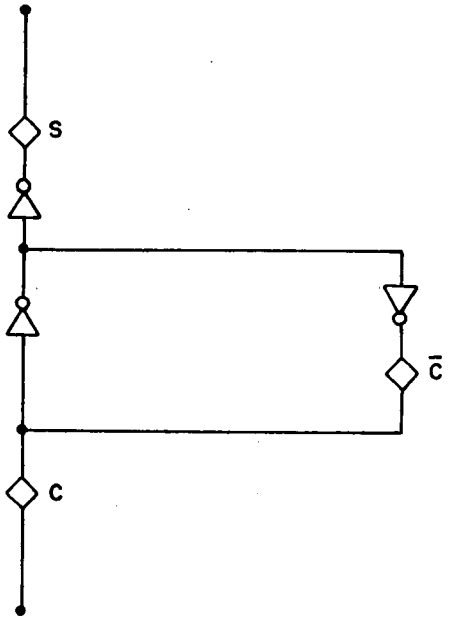
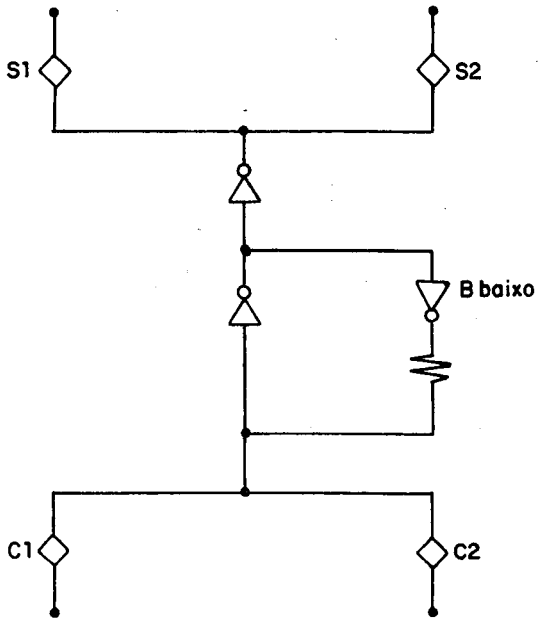


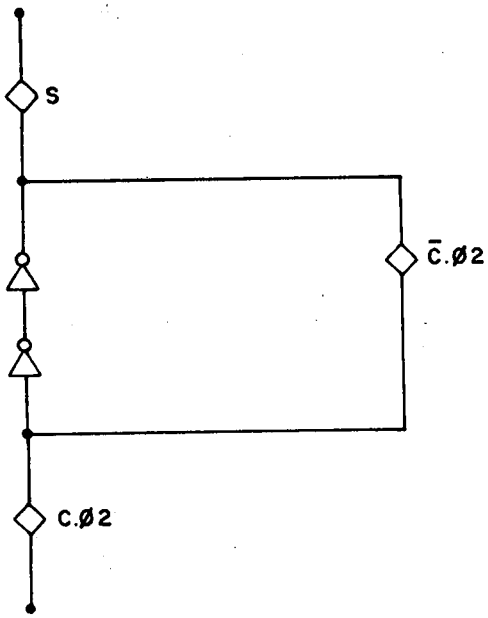
Fig. A3.2 - Sincronizações: a)2 fases não coincidentes e não superpostas; b)3 fases não coincidentes e não superpostas; c)4 fases não coincidentes e não superpostas; d)2 fases, Ph12 e Ph14 fases falsas.



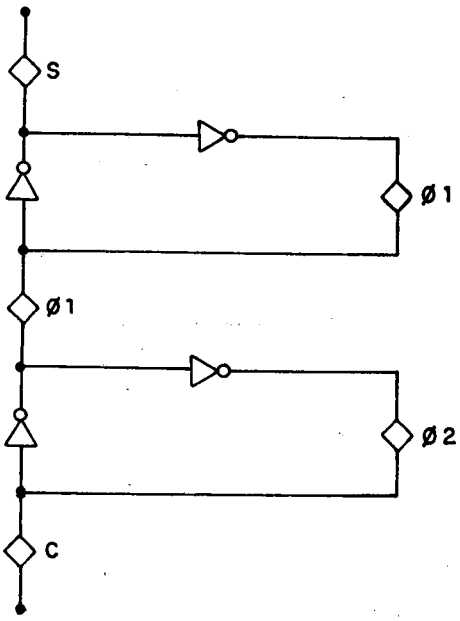
ESTÁTICO SIMPLES



ESTÁTICO ESPECIAL

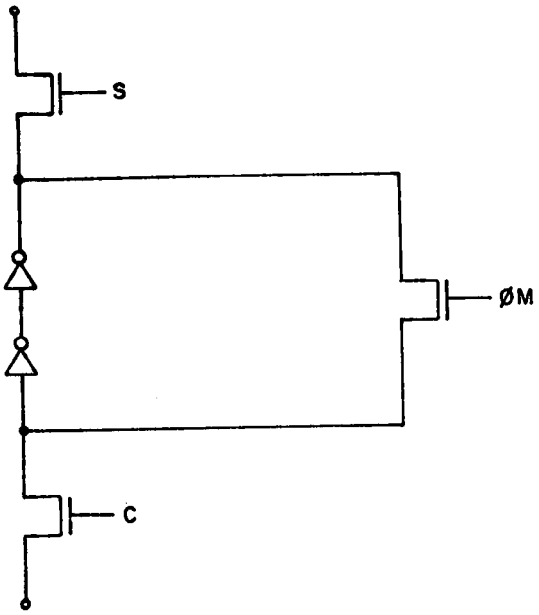


DINÂMICO SIMPLES

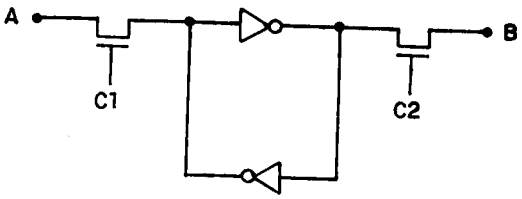


DINÂMICO ESPECIAL

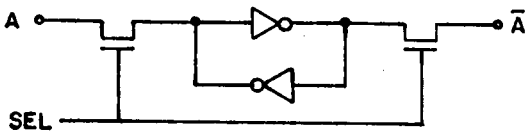
Fig A3.3 - Registradores passíveis de uso.



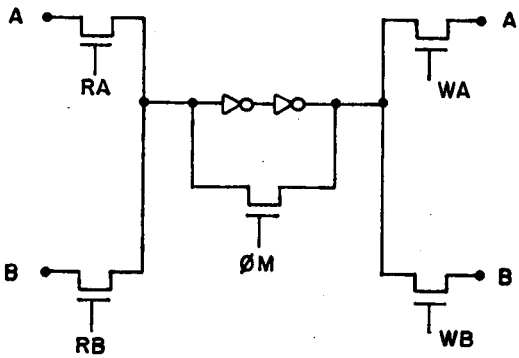
SEMI-ESTÁTICOS



BANCO DE REGISTRADORES
ESTÁTICOS
(CÉLULA ANALÓGICA)



BANCO DE REGISTRADORES
ESTÁTICOS
BARRAMENTO COMPLEMENTADO



BANCO DE REGISTRADORES
SEMI-ESTÁTICOS

Fig A3.3 - registradores passíveis de uso: continuação

ANEXO 4

ENTRADA LEX DO ggcel

```
%%

[a-zA-Z][a-zA-Z0-9]{0,7}      { yylval.sval=(char *)calloc(9,
                                              sizeof(char));
                                strcpy(yylval.sval,yytext);
                                return(NOME);
                              }
[0-9]+                        { yylval.ival=atoi(yytext);
                                return(NUM);
                              }
[0-9]+"."[0-9]* |
[0-9]*"."[0-9]+               {sscanf(yytext,"%f",&yylval.fval);
                                return(ELETRICO);
                              }
[ \t\n]                       { ; /* ignora brancos */
                              }
"/""*".*"*""/"                { ; /*ignora comentarios em 1 linh
                              }
[hvnpdbsmg]             { yylval.cval=yytext[0];
                                return(CARACTER);
                              }
[NMPFCTWBO","""("")""=""[""\]"]          { return(yytext[0]);
                              }
```

ANEXO 5

ENTRADA YACC DO ggcel

```
%{
/*****************************************************************************
        Gerador de Geradores - projeto GPO

        feito por Luigi Carro em 05/10/88

        alterado por Luigi Carro em 27/10/88;

*****************************************************************************/

%}

%start texto     /* simbolo inicial */

%union {
        int ival;
        char *sval;
        float fval;
        char cval;
        }

%token <sval> NOME
%token <ival> NUM
%token <fval> ELETRICO
%token <cval> CARACTER

%type <cval> camada
%type <cval> tipo
%type <cval> orientacao
%type <fval> fator
%type <sval> lista_s lista_e


%%      /* comeco das regras */

texto   :       linha
        |       texto linha
        ;
linha   :       'N' NOME '('lista_s')''('lista_e')'
        |       'M' NUM','NUM
        |       'C' camada NUM','NUM
        |       'B' tipo NUM','NUM
        |       'W' tipo NUM','NUM  NUM','NUM
        |       'F' camada NUM','NUM NUM','NUM fator
        |       'P' camada NUM','NUM NUM','NUM fator
```

```
        |       'T' tipo orientacao NUM','NUM','NUM  NUM fator
        |       'O' NOME '=' NUM 'E'
        |       ']'
        |       'F' camada NUM','NUM  NUM','NUM NOME'='ELETRICO
        |       'P' camada NUM','NUM  NUM','NUM NOME'='ELETRICO
        |       'T' tipo orientacao NUM','NUM','NUM  NUM NOME'='ELETRI
        |       error
        ;

lista_s :       /* vazia */
        |       NOME
        |       lista_s ',' NOME
        ;

lista_e :       /* vazia */
        |       NOME
        |       lista_e ',' NOME
        ;

camada  :       CARACTER
        ;

tipo    :       CARACTER
        ;

fator   :       /* vazio */
        |       ELETRICO
        ;

orientacao :    CARACTER
           ;
%%
# include "lex.yy.c"
```

ANEXO 6

ENTRADA LEX DO COMPILADOR DE ENTRADA

```
%%
NAME              {yylval.cval=yytext[0]; return(NAME);}
TECHNO          {yylval.cval=yytext[0]; return(TECHNO);}
BUS               {yylval.cval=yytext[0]; return(BUS);}
PHASE             {yylval.cval=yytext[0]; return(PHASE);}
Nbits            {yylval.cval=yytext[0]; return(Nbits);}

COMP           {yylval.cval=yytext[0]; return(COMP);}
ELECT           {yylval.cval=yytext[0]; return(ELECT);}
CONNEC         {yylval.cval=yytext[0]; return(CONNEC);}
INTERVAL       {yylval.cval=yytext[0]; return(INTERVAL);}
ELEMENTS        {yylval.cval=yytext[0]; return(ELEMENTS);}
REGISTER     {yylval.cval=yytext[0]; return(REGISTER);}
OPERATOR        {yylval.cval=yytext[0]; return(OPERATOR);}
REGOPERA     {yylval.cval=yytext[0]; return(REGOPERA);}
USR             {yylval.cval=yytext[0]; return(USR);}

[ABC]             {yylval.cval=yytext[0];return(CBUS);};

[a-zA-Z]([a-zA-Z0-9"_"]{0,13})      { strcpy(yylval.sval,yytext);
                                      return(NOME);
                                    }
[0-9]+                              { yylval.ival=atoi(yytext);
                                      return(NUM);
                                    }
[0-9]+"."[0-9]* |
[0-9]*"."[0-9]+                     {sscanf(yytext,"%f",&yylval.fval);
                                      return(ELETRICO);
                                    }
[ \t\n]                             { ; /* ignora brancos */
                                    }
"/""*".*"*""/"                      { n_linhas++; /*ignora comentarios
                                                    em 1 linha*/
                                    }

["*""["\]""{"}""("">""","("")""="";":"]  { return(yytext[0]);}

[ES]                          { return(yytext[0]);}
```

ANEXO 7

ENTRADA YACC DO COMPILADOR DE ENTRADA

```
%{

/*****************************************************************
 Compilador para linguagem de entrada do gerador de partes
        operativas.

        Feito em 14 de dezembro de 1988 por Luigi Carro
*****************************************************************/

%}
%start texto    /* símbolo inicial */
%union {
         int ival;
         char sval[15];
         float fval;
         char cval;
        }
%type <cval> barramento

%token <sval> NOME
%token <ival> NUM
%token <fval> ELETRICO
%token <cval> NAME,TECHNO,BUS,PHASE,CBUS,Nbits,COMP,ELECT,CONNEC
%token <cval> INTERVAL,USR,REGISTER,OPERATOR,REGOPERA,ELEMENTS

%%

texto    : linha
         | texto linha
         | error
         ;
linha    : NAME ':' NOME ';'
         | TECHNO ':' NOME ';'
         | BUS ':' barramento ';'
         | PHASE ':' fases ';'
         | Nbits ':' NUM ';'
         | COMP ':' NUM '>' lista_c ';'
         | ELECT ':' NUM  '>' lista_e ';'
         | CONNEC ':' NUM '>' INTERVAL NUM
         | ELEMENTS '<' elementos '>' ';'
         ;

barramento : CBUS
           | barramento ',' CBUS
           ;
```

```
fases    : NOME
         | fases ',' NOME
         ;

lista_c  : NOME '=' NUM
         | lista_c ',' NOME '=' NUM
         ;

lista_e  : NOME '=' ELETRICO
         | lista_e ',' NOME '=' ELETRICO
         ;

intervalos :  '[' NUM ',' NUM ']'
                '<' 'E'':' ent_list
               'S' ':' sai_list '>'
           | intervalos '[' NUM ',' NUM ']'
                '<' 'E'':' ent_list
               'S' ':' sai_list '>'
           ;

ent_list   :              /* lista vazia */
           | CBUS
           | ent_list '|' CBUS
           ;

sai_list   :              /* lista vazia */
           | CBUS
           | sai_list '|' CBUS
           ;

elementos  : rotina '(' NOME ',' l_conn ','l_comp ',' l_elect ')'
           | elementos rotina '(' NOME ',' l_conn ','
                                           l_comp ',' l_elect ')'
           ;

rotina   : REGISTER
         | OPERATOR
         | REGOPERA
         | USR
         ;

l_comp   : COMP ':' NUM
         | '*'
         ;

l_elect  : ELECT ':' NUM
         | '*'
         ;
l_conn   : CONNEC ':' NUM
         |'*'
         ;
```

```
%%

#include "lex.yy.c"
```

ANEXO 8

MANUAL DE UTILIZAÇÃO E DO ADMINISTRADOR

Para chamar o gerador de partes operativas, basta digitar **gpo <nome do arquivo de descrição>**, onde o último é o nome do arquivo onde se encontra a descrição textual da parte operativa. Esta deve obedecer as regras de sintaxe expostas no anexo 7. Caso existam algumas discordâncias, o programa enviará uma mensagem de erro e abortará.

Devido a seu caráter de ferramenta ainda não totalmente profissional, existem alguns cuidados que não foram tomados no compilador, que devem ser objeto de atenção dos usuários iniciais. Em primeiro lugar, é necessário existir no diretório do usuário o arquivo ultima.tec, onde se encontra a última tecnologia utilizada para expansão. O arquivo deve conter os nomes gren.tcn ou cti.tcn exclusivamente.

Pelo fato da expansão realizar-se por uma chamada a um programa shell e não propriamente pelo compilador, antes de começar o trabalho com o gpo é preciso eliminar todos os arquivos com o sufixo .lds existentes no diretório. Caso se desejar mantê-los, não haverá problemas maiores, mas o programa shell expandirá todas as células.lds presentes no diretório, ocasionando um tempo maior de expansão.

No arquivo de entrada, a ordem é

-parâmetros globais;

-elementos da PO.

Se o usuário não especificar o nome da PO, o número de bits e a tecnologia, o programa abortará. Caso os outros parâmetros não existam, não haverá problemas, desde que somente os parâmetros default sejam acessados. Neste caso, a PO é gerada fazendo-se uso apenas do barramento A.

Ao fim de operação existirá no diretório corrente

o arquivo <nome da PO>.mod, que é o arquivo que pode ser interpretado pelo montador simbólico, através do comando **ms <nome da PO>**. Após a montagem, o comando **gr <nome de PO>** gerará as máscaras para a tecnologia contida em ultima.tec, colocando o conjunto de retângulos RS no arquivo PO.cel. Este pode ser chamado para visualização pelos programas **er** ou **me**.

Se for desejada avaliação elétrica, chama-se **gpoavl <nome saída>**, especificando-se o nome do arquivo onde estarão armazenadas as respostas.

MANUAL DO ADMINISTRADOR

O administrador deve tomar conhecimento dos seguintes arquivos que fazem parte do gpo:

-/usr/luigi/compilador/lngent :compilador para a linguagem de entrada;

-/usr/luigi/compilador/lnglex :analisador léxico para a linguagem de entrada;

-/usr/luigi/compilador/GPOvar.g :variáveis globais;

-/usr/luigi/compilador/GPOdef.h :definições;

-/usr/luigi/compilador/GPOtab.h : tabela de conversão entre nome do módulo e rotina que o chama;

-/usr/luigi/compilador/fontes : diretório onde se encontram todas as rotinas geradoras de células. A cada nova rotina gerada pelo ggcel este diretório deve ser atualizado;

-/usr/luigi/compilador/objetos :diretório onde se encontram as rotinas de células já compiladas, prontas para ligação;

-/usr/luigi/compilador/mod_rotinas :arquivo de rotinas geradoras de módulos. Cada novo módulo deve ser incluído neste arquivo;

-/usr/luigi/compilador/ggcel :gerador de gerador-de-células;

-/usr/luigi/avaliador/avl :compilador para linguagem de avaliação elétrica;

-/usr/luigi/avaliador/avllex :analisador léxico para linguagem de avaliação elétrica;

-/usr/luigi/aprendizado/gg :compilador para ggcel;

-/usr/luigi/aprendizado/gglex :analisador léxico para ggcel;

-/usr/lib/GPO/gren.avl :arquivo de regras de tecnologia para avaliação elétrica;

-/usr/lib/GPO/modulos :arquivo com nome dos módulos disponíveis;

-/usr/bin/gpo :código executável do gerador;
-/usr/bin/gpoavl :executável do avaliador;
-/usr/lib/simbolico/gren.tcn :tecnologia simbólica;
-/usr/lib/simbolico/cti.tcn :tecnologia simbólica.

Para implementação de um novo módulo, deve-se primeiro desenhar simbolicamente as células que farão parte do mesmo. Por tradição, nomeiam-se aquelas com letras maiúsculas, para ser mais fácil localizar de onde vem o erro no momento da depuração. Utiliza-se o comando N do editor simbólico para colocação do nome da rotina C que gerará a célula. No caso de parâmetros de composição, utilizam-se as opções K e O (veja-se o manual do editor). Para parâmetros elétricos utiliza-se a opção F. Se o fator for permanente, no campo de nome coloque-se um ponto (.).

Com o ggcel geram-se as rotinas C equivalentes das células. A sintaxe é **ggcel <arquivo entrada> <arquivo saída>**. Geralmente, o nome do arquivo de saída utilizado é idêntico ao nome da célula (em maiúsculas) acrescido do sufixo .c.

Tendo-se em mãos as rotinas, deve-se compilá-las com a opção -c do compilador, de maneira a ser manter a tabela de ligação aberta.

Deve-se escrever agora uma rotina C que chama as células na ordem devida e escreve o texto para o montador. As rotinas busca_I(parâmetro) e busca_F(parâmetro) devolvem o valor do parâmetro consultado, um inteiro ou um float respectivamente. As variáveis e funções seguintes também são globais, e podem ser utilizadas.

-Nelementos e nn: inteiro e string que indicam o número do elemento atual que está sendo gerado, com relação a ordem designada na descrição de entrada do gpo;

-n_mm e mm: inteiro e string que indicam o número do sub-módulo atual sendo gerado; são geralmente zerados ao início de cada rotina geradora de módulo;

-n_cc e cc: inteiro e string que indicam o número da célula atual sendo chamada; são geralmente zerados no início de cada rotina geradora de módulos;

-nome_cell(): rotina que fornece um novo número para n_cc e cc;

-fp_cell e fp_saida: ponteiros para arquivos de célula e de texto para o montador respectivamente.

Para formação de módulos e células, obedece-se às seguintes regras: Uma célula de um bit (célula folha) tem o nome de Cnn_cc, onde nn e cc são descritos acima; se um módulo precisa de várias células horizontais, o agrupamento destas é denominado Enn_mm. A montagem de todas as células verticais e horizontais para formação de um módulo é chamada de Enn_gbl, ou seja, é a célula global.

No caso de ser necessário gerar células diretamente, ou seja, quando não existe uma rotina derivada de um layout simbólico para ela (vide módulo entassinc), utilizam-se células e sub-módulos fantasmas, chamados Cfntnn_cc e Efntnn_mm. Na verdade, toda a nomenclatura é puramente convencional, visto que serve apenas para diferenciar arquivos. A única exigência real é que a célula englobante chame-se Enn_gbl, visto que este é o nome que o compilador procura quando gera a última linha para o montador, agrupando cada módulo um ao lado do outro na ordem em que foram encontrados na entrada.

No caso de dúvidas, as rotinas rAestsimp, entassinc e somest são ótimos exemplos de construção de nomes. Quando existem intervalos verticais em que os comandos são trocados (parâmetros de conexão na linguagem de entrada), existe uma estrutura de dados que é acessada para consulta de parâmetros. Assim, antes de qualquer consulta a busca_I() ou busca_F() deve-se ter o seguinte trecho:

```
p_intervalo= *p_INTV_list;
CONcopia_valores();
```

Um bom exemplo do uso da estrutura de dados encontra-se na rotina rdinesp.

Com a rotina pronta, pode-se fazer um programa C que forneça as rotinas globais utilizadas para teste da rotina isoladamente, visto que a recompilação de todo o

sistema demora alguns minutos (entre 6 e 8, dependendo da carga na máquina). Depois, deve-se chamar o montador para verificar se o layout gerado corresponde ao esperado para toda a gama de parâmetros de composição.

Após a depuração, para inclusão da rotina no sistema faz-se o seguinte procedimento:

-incluir o nome do módulo no arquivo GPOtab.h, ao final da estrutura. O nome entre aspas é o nome com que o módulo é chamado na descrição de entrada do gerador, enquanto que o nome seguinte é o nome da rotina C;

-incluir a rotina do módulo no arquivo mod_rotinas, não esquecendo de atualizar a rotina erros_msg(err), que armazena as mensagens de erro enviadas ao usuário durante o processo de validação de parâmetros;

-incluir o nome do módulo no arquivo /usr/lib/GPO/modulos;

-compilar o arquivo y.tab.c e ligá-lo as rotinas de células (tanto as antigas quanto as novas), com a opção -ll (biblioteca lex);

-trocar o nome de a.out para gpo e colocar este último no diretório /usr/bin. Para este passo será necessária a presença do super-usuário do ED680.

Se for preciso a inclusão de um novo parâmetro elétrico ou de composição, deve-se colocá-lo na estrutura equivalente no arquivo GPOvar.g, procedendo normalmente com a recompilação.

Como se vê, o uso da linguagem C na sua totalidade e de variáveis globais traz ao mesmo tempo liberdade e perigo. A liberdade de todo o poder de uma linguagem de programação de alto nível, e o perigo de um administrador incauto alterar perigosamente uma variável global, como o nome do arquivo de saída por exemplo. Este problema pode ser facilmente sanado nas próximas versões do gpo pela melhor organização do programa, definição de rotinas que escondam a estrutura de dados do programa principal e pela definição de um subset de C (através de macros por exemplo) que sirva apenas para manipulação dos módulos.

O administrador é o encarregado de gerar os trechos das rotinas que produzem o texto para o avaliador elétrico (vide anexos 10 e 11 e o capítulo 6). Para

alteração do arquivo de tecnologia do avaliador, basta trocar os valores de <tecnologia>.avl (vide anexo 9). Ao lado de cada valor encontra-se um pequeno comentário que explica o valor dos parâmetros que, a menos dos coeficientes de equações (descobertos por simulações), são facilmente obtidos dos parâmetros SPICE ou ARAMOS.

ANEXO 9

ARQUIVO DE TECNOLOGIA gren.avl

```
0.024          Rpoly: em kohms                                  $
3.6e-6         Cappoly: em pF/micra quadrada                    $
1.86e-6        Capmetal: em pF/micra quadrada                   $
105.0e-6       Cjn: do transistor n                             $
240.0e-6       Cjsw: do transistor n                            $
86.0e-5        Cox: em pF por micra quadrada                    $
21.3           kp=mobilidade*Cox do transistor p                $
54.34          Kn                                               $
3.0            Lmetal, largura mínima de metal                  $
2.0            Lpoly, largura mínima de poly                    $
4.0            Wmin, mínimo W da tecnologia                     $
2.0            Lmin, mínimo L da tecnologia                     $
2.0            TRdefault, subida do driver de comando           $
2.0            TFdefault, descida do driver de comando          $
1.6            rdrivercomando, Kohm, w/l=12/2                   $
0.1053         A1, levantado por simulacoes                     $
0.1289         A2                                               $
0.1454         B1                                               $
4.4405         B2                                               $
0.1121         D1                                               $
0.0311         D2                                               $
0.1567         F1                                               $
4.5902         F2                                               $
-8.272e-5      K11                                              $
1.0011         K12                                              $
8.7596         K21                                              $
-4.3793        K22                                              $
-3.454e-6      K31                                              $
1.0026         K32                                              $
22.1726        K41                                              $
-4.4358        K42                                              $
1.0695e-4      upa, polinomio de ajuste R50up, x(3)             $
1.3045e-2      upb                                              $
1.2307         upc                                              $
0.5682         upd                                              $
-4.7312e-4     downa, polinpmio de R50down, x(3)                $
4.30e-2        downb                                            $
0.9885         downc                                            $
1.3693         downd                                            $
5.0            Vgs                                              $
0.8            Vtp                                              $
1.15           Vtn                                              $
3.85           Vdsn                                             $
4.2            Vdsp                                             $
```

ANEXO 10

ENTRADA LEX DO AVALIADOR ELETRICO

```
a [a-zA-Z]
A [a-zA-Z0-9"."]
%p 3500
%k 2000
%%
TECNOLOGIA      {yylval.cval=yytext[0]; return(TECNOLOGIA);}
NBITS           {yylval.cval=yytext[0]; return(NBITS);}
CBUS            {yylval.cval=yytext[0]; return(CBUS);}
Cgphi1          {yylval.cval=yytext[0]; return(Cgphi1);}
Cgphi2          {yylval.cval=yytext[0]; return(Cgphi2);}
Cgphi1N         {yylval.cval=yytext[0]; return(Cgphi1N);}
Cgphi2N         {yylval.cval=yytext[0]; return(Cgphi2N);}
VDD             {yylval.cval=yytext[0]; return(VDD);}
GND             {yylval.cval=yytext[0]; return(GND);}
CdBusA          {yylval.cval=yytext[0]; return(CdBusA);}
CdBusB          {yylval.cval=yytext[0]; return(CdBusB);}
CdBusC          {yylval.cval=yytext[0]; return(CdBusC);}
EQUACOES     {yylval.cval=yytext[0]; return(EQUACOES);}
MODULO:      {yylval.cval=yytext[0]; return(MODULO);}
Ccomando     {yylval.cval=yytext[0]; return(Ccomando);}
Ntransistores  {yylval.cval=yytext[0]; return(Ntransistores);}
CAMINHO      {yylval.cval=yytext[0]; return(CAMINHO);}
CRITICO      {yylval.cval=yytext[0]; return(CRITICO);}
INTERVALOS    {yylval.cval=yytext[0]; return(INTERVALOS);}
BARRAMENTO    {yylval.cval=yytext[0]; return(BARRAMENTO);}
MSG          {yylval.cval=yytext[0]; return(MSG);}
TR         {yylval.cval=yytext[0]; return(TR);}
Dwl        {yylval.cval=yytext[0]; return(Dwl);}
RC         {yylval.cval=yytext[0]; return(RC);}
Cg         {yylval.cval=yytext[0]; return(Cg);}
TP         {yylval.cval=yytext[0]; return(TP);}


[ABC]           {yylval.cval=yytext[0];return(BUS);};

{a}({A}{0,18})     { strcpy(yylval.sval,yytext);
                     return(NOME);
                   }
[0-9]+             { yylval.ival=atoi(yytext);
                     return(NUM);
                   }
[0-9]+"."[0-9]* |
[0-9]*"."[0-9]+    {sscanf(yytext,"%f",&yylval.fval);
                     return(ELETRICO);
                   }
```

```
[ \t\n]                         { ; /* ignora brancos */
                                }
"$".*"$"                      { strcpy(yylval.slval,yytext);
                                return(COMENTARIO);
                                }

["*""["\]"{"}"|">","("")"="";":"]     {
                                        return(yytext[0]);}

[npDES]                         { return(yytext[0]);}

"#".*"#"                      { strcpy(yylval.slval,yytext);
                                return(RCTIPO);
                                }
```

ANEXO 11

ENTRADA YACC DO AVALIADOR ELETRICO

```
%{
/* ************************************************************************
        Avaliador de desempenho elétrico

        Feito por Luigi Carro em Jan. de 1989.

        Parte da dissertação Gerador de Partes operativas
************************************************************************ */

%}
%start texto
%union {
        int ival;
        float fval;
        char sval[18];
        char slval[80];
        char cval;
        }
%token <sval> NOME
%token <cval> TECNOLOGIA NBITS BUS CBUS VDD GND Cgphi1 Cgphi2
%token <cval> Cgphi1N Cgphi2N
%token <cval> CdBusA CdBusB CdBusC EQUACOES MODULO
%token <cval> Ccomando Ntransistores
%token <cval> CAMINHO CRITICO INTERVALOS MSG BARRAMENTO
%token <cval> TR TF TP Dwl RC Cg
%token <ival> NUM
%token <fval> ELETRICO
%token <slval> COMENTARIO RCTIPO

%%
texto   : linha
        | texto linha
        | error
        ;

linha   : TECNOLOGIA '=' NOME ';'
        | NBITS '=' NUM ';'
        | Cgphi1  '=' ELETRICO ';'  /* número de w/l pendurados *
        | Cgphi2  '=' ELETRICO ';'  /* número de w/l pendurados *
        | Cgphi1N  '=' ELETRICO ';'  /* número de w/l pendurados
        | Cgphi2N  '=' ELETRICO ';'  /* número de w/l pendurados
        | VDD '=' ELETRICO ';'
        | GND '=' ELETRICO ';'
        | CdBusA '=' ELETRICO ';'
        | CdBusB '=' ELETRICO ';'
```

```
        | CdBusC '=' ELETRICO ';'
        | EQUACOES ':' l_mod
        ;

l_mod   : MODULO  NOME '(' NUM ')' ':'
            '{' l_calc '}' ';'
        | l_mod MODULO  NOME '(' NUM ')' ':'
            '{' l_calc '}' ';'
        ;

l_calc  : Ccomando '=' ELETRICO ';'
          Ntransistores '=' NUM ';'
          l_calc2
        ;

l_calc2 : caminho
        | l_calc2 caminho
        ;

caminho : CAMINHO CRITICO 'n' '>' NUM
          INTERVALOS '{' l_atributos '}' ';'
        | CAMINHO CRITICO 'p' '>' NUM
          INTERVALOS '{' l_atributos '}' ';'
        | CAMINHO BARRAMENTO BUS ':' 'n'
          '{' l_BUSatributos '}' ';'
        | CAMINHO BARRAMENTO  BUS ':' 'p'
           '{' l_BUSatributos '}' ';'
        ;

l_atributos   : termo
              | l_atributos termo
              ;

termo   : TR '(' NUM ')' '=' 'D' ';' /* default */
        | TR '(' NUM ')' '=' ELETRICO ';' /* t em ns */
        | TF '(' NUM ')' '=' 'D' ';' /* default */
        | TF '(' NUM ')' '=' ELETRICO ';' /* t em ns */
        | Dwl '(' NUM ')' '=' 'D' ';' /* default */
        | Dwl '(' NUM ')' '=' ELETRICO ';' /* numero de w/ls */
        | Cg '(' NUM ')' '=' 'D' ';' /* default */
        | Cg '(' NUM ')' '=' ELETRICO ';' /* numero de w/ls */
        | Cg '(' NUM ')' '=' CBUS '(' BUS ')' ';' /* barramento */
        | RC '(' NUM ')' '=' RCTIPO ';'
        |MSG '('COMENTARIO ')' ';'
        ;

l_BUSatributos  : Btermo
                | l_BUSatributos Btermo
                ;

Btermo  : TR  '=' 'D' ';' /* default */
        | TR  '=' ELETRICO ';' /* t em ns */
```

```
        | TF   '=' 'D' ';' /* default */
        | TF   '=' ELETRICO ';' /* t em ns */
        | Dwl  '=' 'D' ';' /* default */
        | Dwl  '=' ELETRICO ';' /* numero de w/ls do driver */
        | TP '=' 'D' ';' /* default */
        | TP '=' ELETRICO ';' /* numero de w/ls do driver */
        | Cg '=' 'D' ';' /* default */
        | Cg '=' ELETRICO ';' /* numero de w/ls */
        | Cg '=' CBUS '(' BUS ')' ';' /* barramento */
        | RC '(' NUM ')' '=' RCTIPO ';'
        | MSG '('COMENTARIO ')' ';'
        ;

%%

#include "lex.yy.c"
```

BIBLIOGRAFIA

/ANC 86/ ANCEAU, F. The architecture of microprocessors. Reading, Addison - Wesley, 1986.

/BAK 86/ BAKER, A. Selecting a Silicon Compiler. VLSI Systems Design, Palo Alto, 7(5):52-9, May 1986.

/BAR 85/ BARTHOLOMEUS, M. et al. PLASCO: a procedural Silicon Compiler for PLA based systems. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE, Portland, May 20-3, 1985. Proceedings. New York, IEEE, 1985. p.384-8.

/BER 87/ BERENBAUM, A. D. et al. CRISP: A 32-bit Microprocessor with 13-kbit of cache memory. IEEE Journal of Solid State Circuits, New York, 22(5):776-82, Oct. 1987.

/BUD 81/ BUDDE, D. L. et al. The execution unit for the VLSI 432 General Data Processor. IEEE Journal of Solid State Circuits, New York 16(5):514-21, Oct. 1981.

/BUR 86/ BURICH, M. R. Design of Module Generators and Silicon Compilers. In: NATO ADVANCED STUDY INSTITUTE ON LOGIC SYNTHESIS AND SILICON COMPILATION FOR VLSI DESIGN. L'Aquila, July 7-18, 1986. Design systems for VLSI circuits. Dordrecht, Martinus Nijhoff, 1987. p. 403-37.

/CAM 85/ CAMPOSANO, R. & WEBER, R. Compilation and internal representation of digital systems specifications in DSL. In: SIMPOSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 2, Porto Alegre, 20-27 Jul. 1985. Anais. Porto Alegre, SBC, 1985. p.207-22

/CAR 88a/ CARRO, L. & SUZIM, A. A. Metodologia em Concepção de Circuitos Integrados. Porto Alegre, PGCC da UFRGS, 1987. 19 p. RP 87

/CAR 88b/ CARRO, L. et al. GROM: Gerador de ROMs parametrizável. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRONICA, 3, São Paulo, 12-14 jul. 1988. Anais. São Paulo, SBC, 1988. p. 471-81.

/CAR 89/ CARRO, L & SUZIM, A. A. GUAPO: Gerador de POs: manual do usuário. Porto Alegre, PGCC da UFRGS, 1989.

/CHA 86/ CHAO, H. H. et al. Micro 370: A 32-bit Single-Chip Microprocessor. IEEE Journal of Solid State Circuits, New York, 21(5):733-40, Oct. 1986.

/CHU 84/ CHUQUILLANQUI BERNAOLA, S. H. Une nouvelle approche pour l'optimisation topologique et l'automatisation du dessin des masques de PLA complexes. Grenoble, Institut National Polytechnique de Grenoble, 1984.

/CLA 73/ CLARE, C. C. Designing Logic Systems Using State Machines. New York, MacGrawn Hill, 1973.

/COR 88/ CORBIN, V. & SNAPP, W. Design Methodology of the Concorde Silicon Compiler. In: Silicon Compilation. Reading, Addison Wesley, 1988. p.406-45.

/DAV 83/ DAVIO, M., DESCHAMPS, J.P., THAYSE, A. Machines Algorithmiques. Lausanne, Presses Polytechnoques Romandes, 1983.

/DAV SD/ DAVIO, M. The design of processing units. Bélgica, UCL, s.d. 58 p.

/DEM 86/ De MAN, H.; RABAEY, J.; SIX, P. CATHEDRAL II: A Synthesis and Module Generation System for Multiprocessor Systems on a Chip. In: NATO ADVANCED STUDY INSTITUTE ON LOGIC SYNTHESIS AND SILICON COMPILATION FOR VLSI DESIGN. L'Aquila, July 7-18, 1986. Design Systems for VLSI circuits. Dordrecht, Martinus Nijhoff, 1987. p. 571-645.

/DEM 87/ De MAN, H. Evolution of CAD tools towards third generation custom VLSI design. REVUE DE PHYSIQUE APPLIQUEE, 22(1):31-45, Jan. 1987.

/DUC 86/ DUCHENE, Ph. et al. Bibliotheque de Macrocellules CMOS 3 micra. Louvain-la-Neuve, UCL-FAI, 1986.

/EIC 86/ EICHENBERGER, P. Fast Symbolic Layout Translations for Custom VLSI Integrated Circuits. Stanford, Stanford University, 1986.

/EVA 85/ EVANCZUK, S. Silicon Compilers: No Automatic Route to Acceptance. VLSI SYSTEMS DESIGN, Palo Alto, 6(11):42-7, Nov 1985.

/FLE 80/ FLETCHER, W. Sequential Machine Fundamentals. In: An Engeneering approach to digital design. Englewood Cliffs, Prentice-Hall, 1980. p.275-334.

/FOR 87/ FORSYTH, M. et alii. A 32-bit VLSI CPU with 15-MIPS peak performance. IEEE Journal of Solid State Circuits, New York 22(5):768-75, Oct. 1987.

/FRI 85/ FRIEDMAN, E. et al. Parameterized Buffer Cells Integrated into an Automated Layout System. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE. Portland, May 20-3, 1985. Proceedings. New York, IEEE, 1985. p. 389-92.

/FRI 89/ FRIGERI, A. H. FISTLAN - Uma linguagem para descrição de sistemas digitais. Porto Alegre, PGCC de UFRGS, 1989. Relatório de pesquisa a ser publicado.

/GAJ 86/ GAJSKI, D. D. et al. Silicon Compilation In: CUSTOM INTEGRATED CIRCUITS CONFERENCE, Rochester, May 12-15, 1986. Proceedings. New York, IEEE, 1986. p. 102-10.

/GAJ 88/ GAJSKI, D. D. Silicon Compilation. Reading Addison Wesley, 1988.

/GLA 85/ GLASSER, L. A. & DOBBERPUHL, D. W. Circuit Techniques. In: The Design and Analysis of VLSI Circuits. Reading, Addison-Wesley, 1985

/GOM 88/ GOMES, R. F. DARC: um verificador de regras de projeto de CIs utilizando programação em lógica. In: SIMPOSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 3, Gramado, 13-15 abr., 1988. Anais. Gramado, SBC, 1988 p. 85-94.

/GRA 83/ GRANACKI, J.J. & PARKER, A.C. The effect of register-tranfer design trade-offs on chip area and performance. In: DESIGN AUTOMATION CONFERENCE, 20, Miami Beach, June 27-29, 1983. Proceedings. New York, ACM/IEEE, 1983. p. 484-9

/HES 89/ Hessler, A. et al. GALOPA II: gerador parametrizável de PLAs. Porto Alegre, PGCC da UFRGS, 1989. Relatório de pesquisa.

/HIT 83/ HITCHCOCK III, C. Y. & THOMAS, D. E. A Method of Automatic Data Path Synthesis. In: DESIGN AUTOMATION CONFERENCE, 20, Miami Beach, June 27-29, 1983. Proceedings. New York, ACM/IEEE, 1983. p. 484-9

/HOR 83/ HOROWITZ, M. A. Timing models for MOS circuits. Stanford, Stanford University/ Integrated Circuits Laboratory, 1983. Internal Report n. SEL83-003.

/HOR 87/ HOROWITZ, M. et al. MIPS-X: A 20-MIPS peak, 32-bit Microprocessor with on-chip Cache. IEEE Journal of Solid State Circuits, New York, 22(5):790-8, Oct. 1987

/HUN 85/ HUNG-FAI, S. L. Generating an ALU Generator. VLSI Systems Design, Palo Alto, 7(11):98-106, Nov. 1985.

/JAM 86/ JAMIER, R.; BEKKARA, N.; JERRAYA, A. The Automatic Synthesis of Data Procesing Sections. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS. New York, Oct. 6-9, 1986. Proceedings. New York, IEEE, 1986. p. 64-7

/JER 86/ JERRAYA, A.; VARINOT, P.; JAMIER, R.; COURTOIS, B. Principles of the SYCO Compiler. In: DESIGN AUTOMATION CONFERENCE, 23, Las Vegas, June 29 - July 2, 1986. Proceedings. New York, ACM/IEEE, 1986. p. 715-721.

/JHO 85/ JHON, C. S.; SOBELMAN, G. E.; KREKELBERG, D. E. Silicon Compilation Based on a Data-Flow Paradigm. IEEE CIRCUITS AND DEVICES MAGAZINE, New York, 1(3):21-8, May 1985.

/JOH 79/ JOHANNSEN, D. Bristle Blocks: A Silicon Compiler. In: DESIGN AUTOMATION CONFERENCE, 16. San Diego, June 25-27, 1979. Proceedings. New York, ACM/IEEE, 1979. p.310-3.

/JOH 84/ JOHNSON W. et al. Micro VAX 32, A 32 bit Microprocessor. IEEE Journal of Solid State Circuits, New York, 19(5):675-81, Oct. 1984.

/JOU 83/ JOUPPI, N. P. Timing Analysis for nMOS VLSI. In:DESIGN AUTOMATION CONFERENCE, 20, Miami Beach, June 27-29, 1983. Proceedings. New York, ACM/IEEE, 1983. p.411-8.

/KAP 86/ KAPLAN, M. & NG, T. Silicon Compilation of a Core Microprocessor. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE. Rochester, May 12-15, 1986. Proceedings. New York, IEEE, 1986. p. 548-51.

/KER 78/ KERNIGHAN, B. W. & RITCHIE, D. M. The C Programming Language. Englewood Cliffs, Prentice-Hall, 1978.

/KIM 84/ KIM, J. H., McDERMOTT, J., SIEWIOREK, D. P. exploiting Domain Knowledge in IC Cell Layout. IEEE DESIGN & TESI, New York, 1(3):52-64, Aug. 1984.

/KOH 70/ KOHAVI, Z. Finite-state Machines. In: Switching and Finite Automata Theory. New York, Tata McGraw-hill, 1970. chapter 9, p.241-79.

/KUU 85/ KUUTTILA, E. J. et al. IC Design Productivity Study. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE, Portland, May 20-3, 1985. Proceedings. New York, IEEE, 1985. p. 153-4.

/LEW 86/ LEWIS, J. A. et al. Integrated Circuit Procedural Language. Hewlett-Packard Journal, Palo Alto, 37(6):4-10, June 1986.

/LIN 87/ LIN, S., GAJSKI, D. TAGO, H. A Flexible-cell Approach for Module Generation. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE, Portland, May 4-7, 1987. Proceedings. New York, IEEE, 1987. p.9-12.

/MAR 86/ MARSHBURN, T. et al. DATAPATH: a CMOS Data Path silicon assembler. In: DESIGN AUTOMATION CONFERENCE, 23, Las Vegas, June 29 - July 2, 1986. Proceedings. New York, ACM/IEEE, 1986. p.722-9

/MAR 89/ MARCHIORO, G. M. & CARRO, L. Implementação de um editor simbólico para máscaras de circuitos integrados. Porto Alegre, PGCC da UFRGS, 1989. Relatório de pesquisa a ser publicado.

/MAT 85/ da MATA, J. M. ALLENDE: Uma linguagem de especificação de layouts de circuitos integrados. In: SIMPOSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 2. Porto Alegre, 20-27 jul. 1985. Anais. Porto Alegre, SBC, 1985. p.145-58

/MEE 88/ MEERSCH, E. V. Knowledge-Based ERC of Full Custom MOS Circuits. In: SIMPOSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 3, Gramado, 13-15 abril, 1988. Anais. Gramado, SBC, 1988. p. 65-72

/MER 84/ MERCER, M. R. & AGRAWAL, V. D. A Novel clocking technique for VLSI testability. IEEE Journal of Solid - State Circuits, New York, 19(2):207-12, Apr. 1984.

/NIE 86/ NIESSEN, C. Abstraction Requirements in Hierarchical Design Methods. In: Design Methodologies. Amsterdan, North-Holland, 1986. p.151-82.

/NOG 85/ NOGETCH, J. T. & HEDGES, T. Automated Design of CMOS Leaf Cells. VLSI SYSTEMS DESIGN, Palo Alto, 7(11):66-78, Nov. 1985.

/OBR 82/ OBRESKA, M. Etude compartive de differentes methodes de conception des Parties de Controle des microprocesseurs. Grenoble Institut National Polytechnique dc Grenoble, 1982.

/OUS 85/ OUSTERHOUT, J. K. A Switch-Level Timing Verifier for Digital MOS VLSI. IEEE Transactions on CAD, New York 4(3):336-49, July 1985.

/PAN 87/ PANGRLE, B. M. & GAJSKI, D. D. Design Tools for Intelligent Silicon Compilation. IEEE Transactions on Computer Aided Design, New York, 6(6):1098-112, Nov 1987.

/PAR 88/ PARK, N. & PARKER, A. Theory of clocking for Maximum execution overlap of hign-speed digital systems. IEEE Transactions on Computers, New York, 37(6):678-90, June 1988.

/PET 85/ PETERSON, J. L. Operating Systems Concepts. Reading, Addison-Wesley, 1985.

/POW 87/ POWELL, S. & BAROUCH, B. MAP: a parameterizable module generator development system oriented to IC designers. In: IEEE CUSTOM INTEGRATED CIRCUITS CONFERENCE, Portland, May 4-7, 1987. Proceedings. New York, ACM/IEEE, 1987. p. 5-8

/PRE 88/ PREAS, B. T. & LORENZETTI, M. J. Physical Design Automation of VLSI Systems. Menlo Park, Benjamin/Cummings, 1988.

/PUT 82/ PUTATUNDA, R. AUTO-DELAY: A program for automatic calculation of delay in LSI/VLSI chips. In: DESIGN AUTOMATION CONFERENCE, 19. Las Vegas, June 14-16, 1982. Proceedings. New York, ACM/IEEE, 1982. p.616-21.

/RAB 85/ RABAEY, J. M. et al. An Integrated Automated Layout Generation System for Digital Signal Processing Circuits. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE. Portland, May 20-23, 1985. Proceedings. New York, IEEE, 1985. p. 217-20

/RAP 87/ RAPP, V.; HOLZL, G. & GEOURGIOU, C. An expert system fuctional description for parameterized cells. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE. Portland, May 4-7, 1987. Proceedings. New York, IEEE, 1987. p.147-52

/ROS 85/ ROSEBRUGH, C. P. & VELLENGA, J. H. Circuit Synthesis for the SILC Silicon Compiler. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE. Portland, May 20-23, 1985.Proceedings. New York, IEEE, 1985. p. 384-8.

/ROW 87a/ ROWSON, J., WALKER, S. & DIHOLAKIAS, S. A Datapath Compiler for Standard Cells and Gate Arrays. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE. Portland, May 4-7, 1987. Proceedings. New York, IEEE, 1987. p. 149-52

/ROW 87b/ ROWSON,J & WALKER,B. Inside a 2901 Data Path Compiler. VLSI SYSTEMs DESIGNER'S, 1987. p.86-93 User's guide to design automation.

/RUE 86/ RUETZ, P. A. et al. Automatic layout generation of real-time digital image processing circuits. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE. Rochester, May 12-15, 1986.Proceedings. New York, IEEE, 1986.

/RUB 83/ RUBINSTEIN, J.; PENFIELD, P. ; HOROWITZ, M. Signal Delay in RC Tree Networks. IEEE Transactions on CAD, New York, 2(3):202-11, July 1983.

/SAK 83/ SAKURAI, T. Approximation of Wiring Delay in MOSFET LSI. IEEE Journal of Solid State Circuits, New York, 18(4), Aug. 1983.

/SAN 85/ SANGIOVANNI-VINCENTELLI, A. L. An Overview of Synthesis Systems. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE. Portland, May 20-23, 1985. Proceedings. New York, IEEE, 1985. p. 221-5

/SEQ 83/ SEQUIN, C. H. Managing VLSI Complexity: an outlook. Proceeedings of the IEEE, New York, 71(1):149-66, Jan. 1983.

/SHO 88/ SHOJI, M. Circuit Performance Evaluation. In: CMOS Digital Circuit Technology. Englewood Clifs, Prentice - Hall, 1988. chapter 6, p.258-286.

/SHR 82/ SHROBE, H. E. The Data Path Generator. In: CONFERENCE ON ADVANCED RESEARCH IN VLSI. Cambridge, Jan. 25-27, 1982. Proceedings. Jr. Dedham, Artech House, 1981. p. 175-81

/SIN 87/ SINGER, M. A Semistandard way to Differentiate Designs. ESD: The Electronic System design Magazine, Boston, 17(11):89-94, Nov. 1987.

/SIS 82/ SISKIND, J. M.; SOUTHARD, J. R. & CROUCH, K. W. Generating Custom High Performance VLSI Design from Succinct Algorithmic Descriptions. In: CONFERENCE ON ADVANCED RESEARCH IN VLSI. Cambridge, Jan. 25-27, 1982. Proceedings. Jr. Dedham, Artech House, 1981. p. 28-39

/SMI 86/ SMITH, K. F. & GU, J. KD2: An Intelligent Circuit Module Generator. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS. New York, Oct. 6-9, 1986. Proceedings. New York, IEEE, 1986.

/SUZ 81/ SUZIM, A. A. Etude des parties opératives à elements modulaires pour processeurs monolithiques. Grenoble, Institut National Polytechnique de Grenoble, 1981.

/SUZ 85/ SUZIM, A. A. Partes Operativas CMOS. In: SIMPOSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 2. Porto Alegre, 20-7 julho, 1985. Anais. Porto Alegre, SBC, 1985. p. 197-206

/SUZ 88/ SUZIM, A. A. A CAD Frame for VLSI Design. In: SIMPOSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 3. Gramado, 13-15 abr., 1988. Anais. Gramado, SBC, 1988. p. 129-45

/TOD 86/ TODESCO, A. R. W. Concepção de um circuito integrado do tipo processador com conjunto de instruções reduzido. Porto Alegre, PGCC da UFRGS, 1986. Dissertação de mestrado.

/TRU 86/ TRULLEMANS, C. Organization et architecture des systemes integres. Louvain-la-Neuve, Bélgica, UCL, 1985.

/TSE 87/ TSENG, C.-J. & SIEWIOREK, D. P. Automated Synthesis of Data Paths in Digital Systems. IEEE Transactions on CAD, New York, 5(3):379-95, July 1986.

/VLS 86/ VLSI SYSTEMS DESIGN'S. New York, 1986. Semicustom design guide.

/WAG 87/ WAGNER, F. R. KAPA - Uma linguagem para a descrição de hardware no nível de Transferência entre Registradores. Porto Alegre, PGCC da UFRGS, 1987. RP-068.

/WAT 87/ WATKINS, M. L. Software Architecture and the Unix Operating System: An Introduction to Interprocess Communication. Hewllet-Packard Journal, Palo Alto, 38(6):26-36, June 1987.

/WES 81/ WESTE, N. H. E. MULGA - An Interactive Symbolic Layout System for the Design of Integrated Circuits. The Bell System Technical Journal, New York, 60(6):823-58, July-Aug. 1981.

/WES 85/ WESTE, N. H. & ESHRAGUIAN, K. Circuit Characterization and Performance Estimation. In: Principles of CMOS VLSI Design. Reading, Addison - Wesley, 1985.

/WIL 82/ WILLIAMS, T. & PARKER, K. Design for Testability - a survey. IEEE Transactions on Computers, New York, 31(1):2-15, Jan. 1982.

/WIL 84/ WILLIAMS, J. S. et al. MEGACELL - A design System for CMOS VLSI. In: EUROPEAN SOLID-STATE CIRCUITS CONFERENCE, 10. Edinburgh, Sep. 1984. Proceedings. Edinburgh, CE Consultants, 1984. p. 127-31

/WOO 86/ WOOD, G. & LAW, S. SKILL-An Interactive Procedural Design Environment. In: IEEE CUSTOM INTEGRATED CIRCUITS CONFERENCE, Rochester, May 12-15, 1986. Proceedings. New York, IEEE, 1986. p.544-7.

/YAS 86/ YASUURA, H. Design and Analysis of Hardware Algorithims. In: Design Methodologies. North Holland, 1986. p.185-214.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Pós-Graduação em Ciência da Computação

Gerador parametrizável de
partes operativas CMOS

Dissertação apresentada aos Srs.

Prof. Dr. Sergio Bampi

Prof. Dr. Raul F. Weber

Prof. Dr. Charles Trullémans

Prof. Dr. Altamiro A. Suzim

Visto e permitida a impressão

Porto Alegre, .11./.04./.89..

Prof. Dr. Altamiro A. Suzim
Orientador

Profa. Ingrid E.S. Jansch Pôrto
Coordenadora do Curso de Pós-Graduação
em Ciência da Computação.

/ZYS 83/ ZYSMAN, E & SUZIM, A. A. Estruturas Microprogramadas para VLSI. In: SIMPOSIO BRASILEIRO DE MICROELETRONICA, 3. São Paulo, 25-7 jul., 1983. Anais. São Paulo, SBC, 1983. v.1, parte 1. p. 59-96