UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CSAR: UM COMPILADOR DE SILICIO VOLTADO A EXECUÇÃO DE OPERAÇöES PARALELAS

por

Rogério Figurelli Gomes

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

Prof. Ricardo Augusto da Luz Reis
Orientador

Porto Alegre, julho de 1990.

CATALOGAÇÃO NA FONTE

Gomes; Rogério Figurelli
CSAR: Um Compilador de Silício voltado à execução de operações paralelas. Porto Alegre, PGCC da UFRGS, 1990.
1v.
Diss. (mestr. ci, comp.) UFRGS-PGCC, Porto Alegre, BR-RS, 1990.

Dissertação: Microeletrônica: PAC de Circuitos Integrados: Compiladores de Silício: Síntese Automática: Geradores de Módulos: Projeto VLSI.

"Todavia, o que mais me preocupa é ver o indivíduo aparentemente seguro de si, fazendo afirmativas habilidosas de que o nada existe e é tão forte que pode criar o mundo."

Silveira Sampaio

Aos meus pais e minha esposa Tania.

AGRADECIMENTOS

A Tania pelo amor e amizade dispensado em todos os momentos.

Aos meus pais, Adroaldo e Neusa, pelo amor e educação recebidos.

Ao professor e amigo Ricardo Reis pela orientação neste trabalho e pelo tempo que trabalhamos juntos.

Aos amigos e colegas de pós-graduação Fernando Moraes, João Netto, Marcelo Lubaszewski, Luis Otávio Freire e Paulo Wagner pela colaboração e amizade.

A Fábio Somenzi e Marcelo Johan pela ajuda e colaboração.

Aos professores Tiarajú Wagner, Rocha Costa e Sérgio Bampi, pela dedicação ao ensino.

SUMÁRIO

LISTA DE ABREVIATURAS

ASIC	Circuitos Integrados de Aplicação Específica
CI	Circuito Integrado
CMOS	Complementary Metal Oxide Silicon
PROLOG	Linguagem de Programação em Lógica
RAM	Memória de Acesso Randômico
ROM	Memória somente de leitura
SYCO	Compilador de Silício desenvolvido em Grenoble
ULA	Unidade Lógica e Aritmética
VLSI	Very Large Scale Integration

LISTA DE FIGURAS

RESUMO

Apesar de o termo Compilador de Silício ser recente, as idéias envolvidas na sua construção são antigas e conhecidas.

Este trabalho apresenta todos os passos para construção do CSAR, um compilador de silício voltado à execução de operaçöes paralelas.

A arquitetura alvo gerada pelo CSAR é discutida, bem como os algoritmos implementados.

ABSTRACT

Although the expression Silicon Compiler be recent, the covered ideas for its building are old and well-known.

This work presents the steps for CSAR building, a silicon compiler apllied to the execution of parallel operations.

The target architecture produced by CSAR is discussed, as well as the implemented algorythms.

1. INTRODUÇÃO

1.1 Apresentação

O grau de dependência tecnológica de um país na área de microeletrônica pode ser avaliado através de uma simples suposição: corte do fornecimento de CIs por outros países. As indústrias de informática e de eletrônica estariam ameaçadas, bem como toda a gama de serviços delas decorrentes ou dependentes. Esta situação de fragilidade é conseqüência da falta de planejamento econômico, com a conseqüente falta de investimentos nas áreas tecnológicas estratégicas.

Consciente da necessidade de uma capacitação nacional em microeletrônica, foi formado um grupo de pesquisa na UFRGS, dedicado à formação de recursos humanos e desenvolvimento de pesquisas na área de microeletrônica [REI87a],[SUZ86]. Uma das áreas de pesquisa mais atuante do GME ou Grupo de Microeletrônica da UFRGS é a síntese automática ou projeto automático de circuitos integrados.

Procura-se neste trabalho desenvolver uma técnica automatizada de projeto de CIs, com aplicações bastante específicas. Pode-se afirmar que quanto menos automatizado o projeto, menor a probabilidade de que funcione perfeitamente. A automação do processo de concepção fornece ao projetista ferramentas de CAD que podem simular o comportamento físico dos dispositivos presentes no CI, tornando muito mais rápido e confiável o projeto [AKI79],[WIL83].

A experiência em projetos ASIC é fundamental para conhecer-se os passos de projeto de um CI e formalizá-los em um programa de computador. Estes passos de projeto são

analisados e detalhados, assim como seu transporte para rotinas e algoritmos computacionais.

Resumindo, pode-se dizer que as duas motivações básicas deste trabalho são:

- contribuir com alternativas e meios de projetar circuitos integrados.

- transmitir a experiência de projetos de CIs ASIC, bem como mostrar caminhos para sua automatização.

1.2 Objetivos

A ênfase principal deste trabalho está no capítulo IV, ou seja, na escolha e estudo de uma arquitetura alvo do Compilador de Silício e de como gerá-la automaticamente. Relacionar estas técnicas com as técnicas tradicionais de síntese automática também é escopo deste trabalho, embora as técnicas adotadas estejam fortemente ligadas ao projeto tradicional de circuitos ASIC.

Procura-se desenvolver um Compilador de Silício que possua as seguintes características:

- Parametrização e Independência de Tecnologia.

- Maior paralelismo nas operações, satisfazendo restrições quanto à velocidade final do circuito e área final a ser ocupada.

A tarefa de desenvolver um Compilador de Silício completo é complexa e exige recursos humanos especializados em diversas áreas, tais como física de semicondutores, engenharia de projeto, algoritmos computacionais, etc. Não é objetivo deste trabalho implementar uma ferramenta completa, com todos recursos encontrados em ferramentas profissionais. Sabe-se que uma ferramenta completa exige o trabalho e dedicação simultânea de diversos profissionais.

O objetivo principal é apresentar e discutir novas metodologias de projeto, bem como elaborar uma seqüência lógica de desenvolvimento de compiladores de silício, ou seja, propor alternativas de ataque ao problema. Entretanto muitos dos algoritmos envolvidos foram implementados e testados, e são devidamente documentados, como os algoritmos de compilação da linguagem de entrada, posicionamento e roteamento e representação simbólica.

Com a escolha de projeto de compiladores de silício deve-se associar os seguintes objetivos:

- velocidade de desenvolvimento de projetos com a obtenção de ciclos de projeto mais curtos,

- obtenção de menos erros e diminuição dos custos de projeto,

- facilidade de uso para projetistas de sistemas.

2.PROJETO AUTOMATICO DE CIRCUITOS INIEGRADOS

2.1 Introducão

Este capítulo apresenta metodologias de projeto de circuitos integrados, e tem por objetivo introduzir a síntese automática de circuitos integrados.

São descritos os níveis de hierarquia presentes no projeto de um CI. Os Compiladores de Silício são apresentados como uma solução de projeto descendente, que partem de um nível abstrato de descrição do circuito para atingir o nível de máscaras.

2.2 Metodologias de Projeto

Os primeiros projetos de CIs eram extremamente manuais. O projetista dispunha de algumas ferramentas para edição de máscara e teste. Era então necessário projetar células ou conjunto de células até compor um bloco qualquer. A comunicação entre os blocos era feita através de rotas projetadas manualmente também [MEA80]. A montagem do circuito era efetuada através do posicionamento dos blocos, formando a planta baixa do circuito.

Pode-se ver na fig.2.1 alguns exemplos das máscaras destes circuitos. Pode-se notar áreas extremamente densas e heterogêneas, caracterizando técnicas manuais de projeto. As áreas mais densas são ocupadas por estruturas regulares como PLAs, ROMs e RAMs.

Entretanto, com o aumento da complexidade dos circuitos integrados tornou-se essencial a automatização de algumas etapas do projeto [NEW87],[SEQ83]. Com o uso de ferramentas de CAD como Verificadores de Regras de Projeto, Extratores Elétricos e Roteadores de Canal, a confiabilidade e velocidade de realização de projetos aumentaram, mostrando a eficiência e utilidade destas ferramentas. Entretanto o projeto até aqui é essencialmente ascendente, ou seja, embora exista uma etapa inicial que determina quais blocos farão parte do circuito final, começa-se projetando células, e juntando-as até completar o circuito. As características básicas desta metodologia dedicada são o alto tempo de projeto e a alta densidade local dos blocos que compõem o circuito, principalmente devido ao fato de o projeto manual das células ainda ser mais eficiente que as técnicas automáticas hoje existentes, e as técnicas de planejamento topológico.

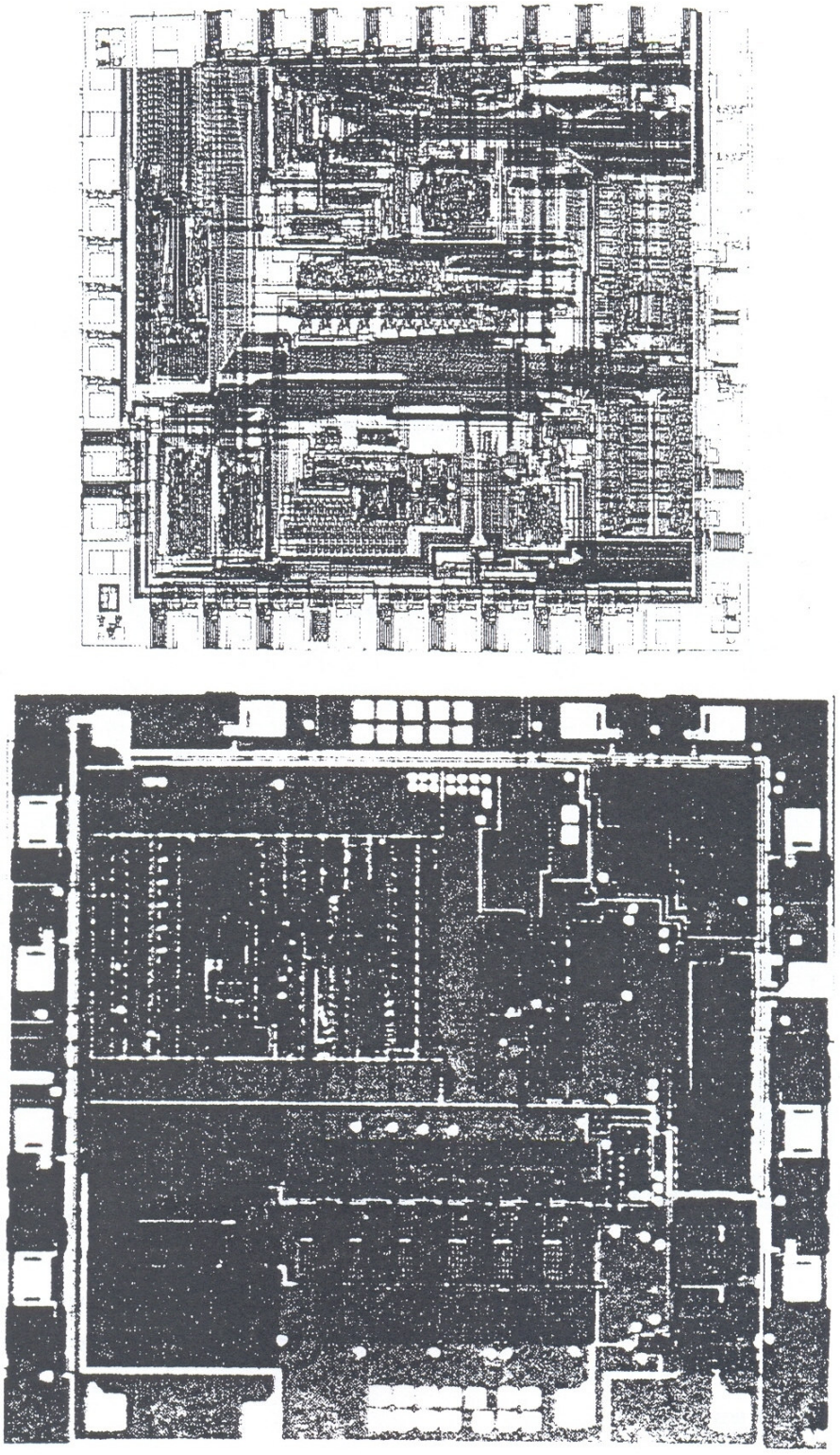


Figura 2.1 Máscaras de dois circuitos NMOS

É discutível, entretanto, a otimização do posicionamento de blocos e do roteamento de canais realizados manualmente ou de forma semi-automática. As técnicas automáticas atuais conduzem geralmente a soluções bem mais otimizadas [THO81]. Entretanto a metodologia ascendente ainda é bastante utilizada, auxiliada por posicionadores e roteadores automáticos. Esta metodologia inicial foi inevitável, pois foi necessário conhecer e dominar os princípios básicos dos projetos de CIs.

Com a diversificação do projeto de células e o aumento do número de projetos, a idéia de reaproveitar células em outros projetos tornou-se bastante viável, através da organização de uma biblioteca de células. Boa parte das metodologias atuais aproveitam bastante da idéia de existência de uma biblioteca de células [REI87b],[REI88]. Uma biblioteca de células deve ser devidamente documentada e simulada, com células que podem ser utilizadas em diversos projetos. Com isso, também é possível realizar projetos numa estratégia ascendente, ou seja, como já existe um conjunto pronto de células projetadas, basta criar regras de alto nível que selecionam quais células devem ser utilizadas, posicionadas e roteadas. Esta metodologia, conhecida por "standard-cell" [REI87b], foi uma das primeiras tentativas de automação das etapas de projeto, e não simplesmente criar ferramentas de CAD que auxiliassem o projetista a realizar algumas etapas do projeto.

Assim como a metodologia "standard-cell", as estratégias de projeto atuais estão tendendo cada vez mais para metodologias descendentes [REI83]. Uma característica destes circuitos é sua regularidade e modularidade. Pode-se ver na fig.2.2 circuitos "standard-cells" e circuitos gerados utilizando geradores de módulos.

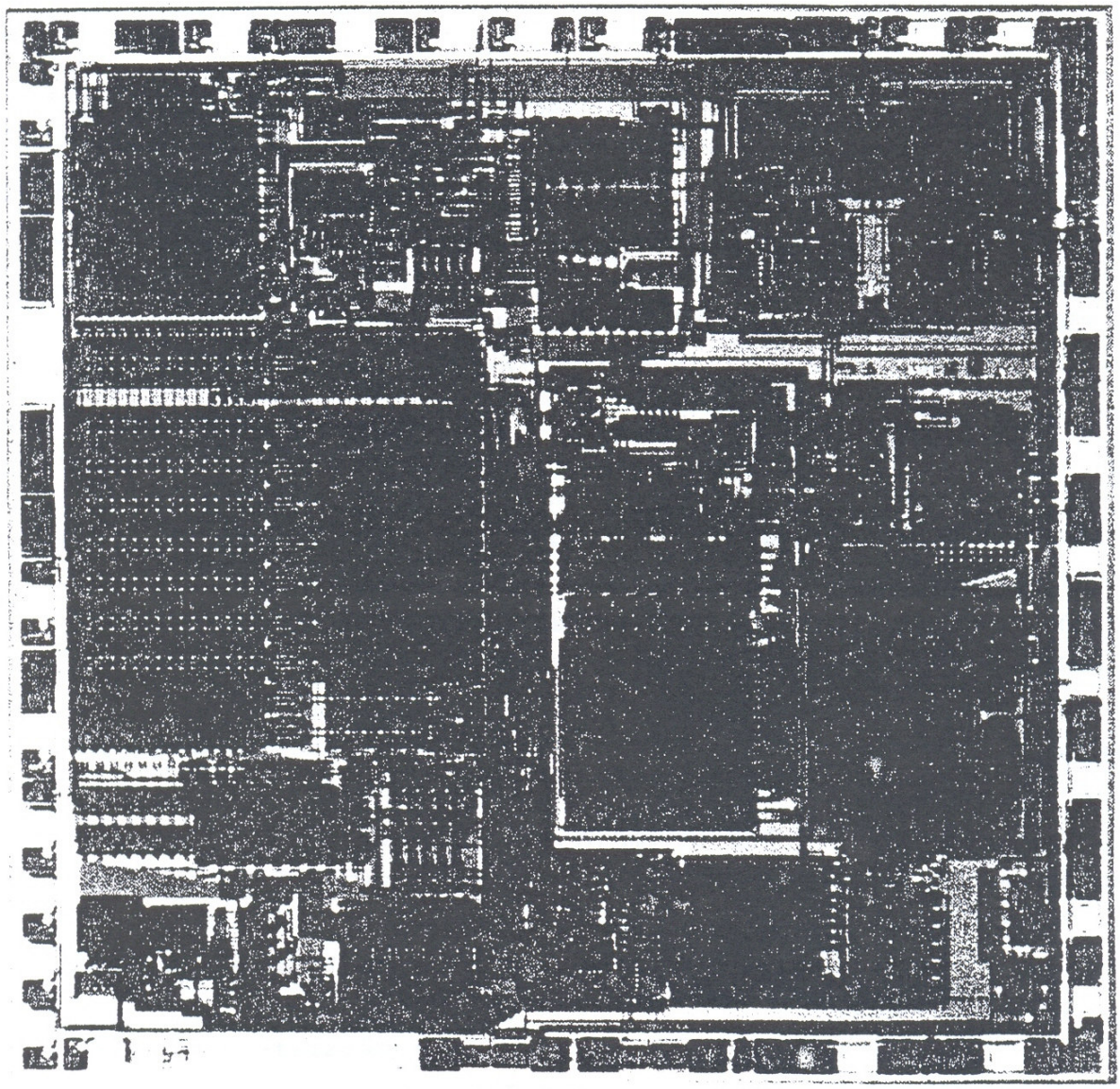


Figura 2.2 Circuitos CMOS que utilizam Síntese Automática

A comparação com os primeiros CIs da fig.2.1 evidencia as novas técnicas de projeto.

2.3 Síntese Automática

O projeto descendente ou ascendente pode ser dividido em diversas etapas, não necessariamente coincidentes. Estas etapas caracterizam os níveis de projeto. Assim como os programas possuem vários níveis, alguns mais próximos da linguagem da máquina (computador) e outros da linguagem do projetista, pode-se obter diversas linguagens para representarmos os diversos níveis de interpretação de um circuito [SUZ81],[CAR87].

A linguagem pode descrever um conjunto de retângulos que representa as máscaras finais do circuito ou um conjunto de portas lógicas e equações booleanas, etc. O importante é que a existência de uma linguagem que expressa o comportamento final do circuito, em alguma etapa do projeto, caracteriza um nível do projeto, ou seja, cada nível é caracterizado por uma linguagem de descrição.

A divisão do projeto em níveis apresenta as seguintes vantagens, no que refere-se à síntese automática:

- permite a divisão do problema, para que seja resolvido em etapas [HAF82].

- permite que algoritmos que façam a compilação de uma linguagem de um nível superior para outro nível inferior possam ser utilizados para diversos projetos de ferramentas de síntese automática, isto é, ferramentas diferentes podem ter etapas comuns.

A passagem de um nivel superior ou mais abstrato para um nível inferior não implica em uma única soluçåo. Quando deseja-se calcular mentalmente o resultado da multiplicaçåo de 10 vezes 3, pode-se realizar o cálculo de diversas formas. Simplesmente colocar um 0 à direita do 3 ou somar 10 + 10 + 10, etc. Da mesma forma, pode-se compilar uma linguagem e implementá-la de diversas formas, conforme as características desejadas por diferentes projetistas.

Quanto mais alto o nível de projeto, mais significativas serão as decisöes tomadas, como acontece, por exemplo, em uma empresa. Se o operador de uma máquina resolver trocar a seqüência de operaçöes da máquina, mesmo que os resultados não sejam bons a empresa terá condiçöes de assimilar as conseqüências. Entretanto se o diretor da empresa tomar uma decisão estratégica errada a estabilidade desta pode ser comprometida. Portanto, a medida que são criadas linguagens que descrevem níveis superiores e que são desenvolvidos compiladores que conduzem a um nível menos abstrato qualquer (máscaras ou lógico, por exemplo), é necessário que diferentes soluçöes sejam estudadas e que sejam escolhidas as que se adaptam melhor às especificaçöes dos circuitos a serem projetados [JEN82]. A escolha de uma arquitetura alvo facilita a busca de soluçöes para a compilaçåo de linguagens, como será estudado no Capítulo IV.

A automatizaçåo do projeto de um CI depende de duas questöes principais:

- escolher quais as etapas ou níveis intermediários devem existir, bem como as linguagens que representam estes níveis.

- criar algoritmos que façam a compilação de uma linguagem para outra.

A necessidade de melhorar o rendimento dos projetos e de diminuir o tempo de concepção conduz ao desenvolvimento de ferramentas de síntese automática. Além disso a confiabilidade do projeto aumenta muito. Um bom exemplo são os geradores de módulo, que geram estruturas bastante regulares. O projetista não precisa se preocupar em projetar uma RAM ou um PLA. Basta especificar seu comportamento e a topologia desejada. A área final pode ser diminuída utilizando-se algoritmos de otimização topológica (como "folding" [WES85], por exemplo).

As soluções obtidas dependem bastante da experiência do projetista, que foi formalizada no programa de síntese. As combinações possíveis são enormes. Gerar automaticamente uma célula entretanto está se tornando viável, com o avanço das pesquisas na área (ver [MOR89], [FEL76]). As células desenhadas manualmente apresentam um sério inconveniente: a dependência tecnológica. As regras de projeto não são necessariamente compatíveis entre os fabricantes (Silicon Foundry) ou usinas de fundição. Além disso, existem diversos tipos de processos, com características diferentes.

O que se faz para contornar o problema de dependência de tecnologia é:

- Em "standard cells" deve-se manter atualizada a biblioteca a cada mudança nas regras de projeto, e, quando necessário, prever células para outras tecnologias e processos.

- Nos geradores de módulos e compiladores de silício, pode-se projetar células em um nível simbólico [GAJ85],[GRA79]. Este nível simbólico é bastante parecido com o nível de máscaras final. O nível simbólico possui valores de regras e fatores de mérito dos transistores parametrizados (este assunto será detalhado no Capítulo V). Sem dúvida a segunda soluçåo aparenta ser a melhor, quanto a este aspecto. A tendência é que as ferramentas que compilam a linguagem simbólica para a linguagem de máscaras (no caso do GME, a linguagem RS [TOD85]) forneçam soluçöes bastante próximas da manual, ou até melhoradas, com a utilizaçåo de compactadores [SHI83]. Neste ponto é interessante notar que dois níveis de descriçåo já foram citados: máscaras e simbólico. Supondo a existência de um terceiro nível, o nível elétrico, onde sejam descritos apenas transistores ou o "net-list" do circuito (como uma descriçåo SPICE [WES85]), é possível fazer a compilaçåo direta do nível elétrico ao nível de máscaras? A utilizaçåo de um nível intermediário, como o simbólico, facilita ou melhora alguma coisa?.

As respostas para estas perguntas devem auxiliar a resolver dúvidas em níveis mais abstratos. A partir do "net-list", é possível posicionar os transistores aleatoriamente e utilizar após um programa de roteamento de conexoes. Algumas regras devem ser respeitadas, como distância entre camadas e largura mínima destas para o roteamento. Enfim, após uma série de aplicaçöes de regras pode-se atingir uma soluçåo satisfatória. Esta soluçåo pode ser a que possui menor área, por exemplo, ou a que melhor se adapta as restriçöes topológicas do circuito global. Existem outras soluçöes para mapear diretamente do nível de transistores para o nível de máscaras, como o caso da metodologia "gate-matrix" [MOR89], e que nåo necessitam do nível simbólico.

Deseja-se agora utilizar o nível simbólico, ou seja, a partir do nível elétrico atingir o nível simbólico e após estes o nível de máscaras. A compilação do nível elétrico para o nível simbólico deve ser mais simples do que diretamente para o nível de máscaras, pois não existem preocupações com muitas das regras de projeto. A compilação do nível simbólico para o nível de máscaras pode utilizar uma ou diversas ferramentas que gerem diversos "layouts" para diversas tecnologias.

Comparando as duas soluções discutidas, chega-se as seguintes conclusões:

-algumas etapas intermediárias ou níveis hierárquicos do projeto global do circuito integrado podem ser evitadas, pois as ferramentas podem ser poderosas o suficiente para conter um grande campo de ação. Deve-se ter consciência do ponto até onde é vantajoso dividir o processo de síntese automática em níveis hierárquicos. Além disso as etapas intermediárias podem ser explícitas ou implícitas na ferramenta.

-a adoção de etapas intermediárias ou níveis hierárquicos torna o sistema mais aberto e mais genérico [HIT83].

De qualquer forma, a escolha das etapas intermediárias não é simples. As etapas intermediárias determinam a compatibilidade da ferramenta com outras ferramentas de síntese automática. O estudo dos níveis de descrição mais conhecidos e normalmente utilizados em diversos projetos é importante. A classificação das ferramentas de síntese dependerá também deste estudo. Em [GAJ86] e [GAJ84] encontra-se um estudo de diferentes níveis de projeto de circuitos integrados, desde um nível mais abstrato, como um algoritmo, até o nível de máscaras. Os níveis, neste caso,

são representados como três eixos tridimensionais. Os níveis de descrição estão associados a três eixos: eixo comportamental, eixo estrutural e eixo geométrico.

Quando o projeto se encontra em um determinado nível, este pode ser representado como um ponto dentro do espaço tridimensional. A ligação entre os diversos pontos que representam os diversos níveis forma um caminho que representa a síntese de um nível mais abstrato para um nível menos abstrato. A forma de apresentar os dados ou resultados da síntese deve estar próxima do eixo geométrico. A síntese lógica parte de um nível algorítmico (comportamental) até um nível lógico (geométrico). A técnica "standard cell" parte de um nível lógico (estrutural), como uma descrição de portas lógicas para obter o nível de máscaras (geométrico). Os Compiladores de Silício ideais partem de um nível algorítmico (comportamental) até um nível de máscaras (geométrico) [GOL85].

O caminho inverso, que parte de níveis menos abstratos até níveis mais abstratos, conhecido como captura também possui as mesmas características. O Diagrama de Gajski serve muito bem como uma demonstração didática do que é buscado nos processos de síntese automática. Porém a aplicação prática destes diagramas pode levar a caminhos diferentes. Na verdade não existe na prática nada que obrigue uma ferramenta de síntese definir claramente em que nível de abstração se encontra a cada instante do processo de síntese.

Resumindo, os objetivos das ferramentas de sínteses são os mesmos, ou seja, partir de níveis o mais abstratos possíveis (portanto o mais longe da origem dos eixos tridimensionais e o mais próximo possível da linguagem

natural) até obter o nível de máscaras. Porém as diferentes formas de alcançar este objetivo tornam as soluções com características bastante diferentes, existindo um campo fértil para pesquisas.

2.4 Compiladores de Silício

Um circuito integrado pode ser definido como um conjunto de retângulos de diversas camadas (ou cores, se preferir) com tamanho e posição definidos de forma precisa.

Sabendo-se que a representação de um retângulo pode ser feita da seguinte forma:

Retângulo = Camada + Posição(x,y) + Tamanho(Dx,Dy)

Pode-se afirmar que projetar um CI é realizar:

I) Somatório Ri = (Ci, Xi, Yi, Dxi, Dyi) para 1 <= i <= N.
onde Ci = camada ou cor do retângulo i.
Xi = posição x do retângulo i.
Yi = posição y do retângulo i.
Dxi = largura do retângulo i.
Dyi = altura do retângulo i.

Com base nesta conclusão, em princípio, um programa gerador de circuitos integrados pode ser feito se os elementos dos retângulos Ri forem gerados aleatoriamente. Deste modo, seria possível gerar uma infinidade de CIs, com diversos tamanhos e números de retângulos?

Analisando o problema por este lado, a solução parece bastante simples. Isso porque uma equação importante não foi

considerada. Esta equaçåo é que torna a tarefa de projetar CIs e de gerá-los automaticamente uma tarefa trabalhosa e complexa. A equaçåo que falta chama-se atendimento as especificaçöes. Ou seja, um conjunto de retângulos não podem ser considerados um CI, se não atenderem uma especificaçåo.

Para que se considere as equaçöes anteriores como verdadeiras é necessário que elas atendam a uma especificaçåo, ou seja, cumpram a funçåo para que foram projetados.

Sendo assim, pode-se agora afirmar com segurança que um CI é:

I) Uma Especificaçåo.
II) Somatório Ri = (Ci, Xi, Yi, Dxi, Dyi) para 1 <= i <= N.
III) Somatório Ri Atende à Especificaçåo.

Tentar resolver o problema de projetar CIs desta forma vai mostrar outra realidade: embora a linguagem final ou forma de implementaçåo de um CI é um conjunto de retângulos, a especificaçåo está bastante longe do formato de retângulos.

De qualquer forma, para gerar automaticamente um CI a partir das equaçöes I, II e III existe no mínimo duas soluçöes:

Solucao 1) Encontra-se uma especificaçåo de um CI a nível de retângulos, ou um programa que, a partir de uma especificaçåo de alto nível, gere uma especificaçåo a nível de retângulos.

Solucao 2) Encontra-se uma estrutura de dados de alto nível que atenda a especificação e cria-se um programa que resolva a equação II.

A primeira solução certamente é extremamente complexa, e leva a uma equação complicada, devido ao conjunto de variáveis que envolve. Determinar todas estas variáveis já é um problema complexo.

A segunda solução chama-se Compilador de Silício.

Os Compiladores de Silício são ferramentas de CAD que fazem a compilação de um nível algorítmico para o nível de máscaras ([GAJ85],[GAJ86]), passando por diversos níveis intermediários. O processo está baseado nos compiladores de linguagens de computadores. A idéia básica é construir ferramentas que permitam ao programador ou usuário abstrair o máximo de detalhes possíveis do programa final desejado. Deve-se manter, por outro lado, o comportamento desejado.

Assim como detalhes da arquitetura do computador em que está trabalhando, podem ser desconsiderados por um programador de uma linguagem de alto nível (se este possuir uma biblioteca considerável de rotinas ou procedures), o projetista pode projetar circuitos esquecendo o nível de máscaras, detalhes de processo e até mesmo outros níveis-intermediários.

O nível de máscaras está no nível mais baixo de concepção de um CI para o projetista. Porém as máscaras estão longe ainda do CI a ser produzido. O projetista não necessita prever os passos típicos para realização dos processos de fabricação. Ele fornece apenas uma descrição textual dos retângulos das diferentes camadas que compõem o circuito. Na

verdade existe um programa ou conjunto de procedimentos que traduzem a linguagem no nível de máscaras fornecida pelo projetista para as máscaras físicas correspondentes.

Algumas das etapas do processo de concepção de um CI (tecnologia CMOS), podem ser descritas com uma linguagem que descreve procedimentos operacionais, por exemplo:

```
{ Produz CI de teste }

prepara silício;
monta wafer;
prepara litografia;
posiciona filme;
.
.
.
teste;
corta wafer;
realiza encapsulamento;
fim;
```

Pode-se ver que o fato de o projetista utilizar apenas uma descrição das máscaras em retângulos geométricos já se constitui em separar o processo total em duas etapas: a etapa de concepção e a etapa de fabricação. O projetista não pode interferir na etapa de fabricação. Porém o fabricante garante que, se a etapa de concepção seguir as regras de projeto, a etapa de fabricação estará correta.

O nível de máscaras gerado pelo projetista é bem mais abstrato (desconsidera detalhes) que o nível do fabricante, porém mantém as informações necessárias para que este nível seja atingido.

Desenvolver um Compilador de Silício é trabalhar para que níveis mais abstratos possam ser obtidos, de maneira a facilitar a ação do projetista. O que se faz atualmente é tentar automatizar a redução de um nível de descrição de um circuito integrado para um nível inferior de descrição, desenvolvendo ferramentas de CAD com este propósito. Para que isso seja possível é necessário que sejam conhecidas e dominadas as etapas feitas normalmente manualmente, ou, em outras palavras, a formalização do conhecimento humano que realiza este procedimento em um conjunto de regras estratégicas. Esta tradução pode gerar um algoritmo simples ou um algoritmo complexo, que exige meios mais formais de representação e técnicas mais apropriadas que escrever um algoritmo simplesmente.

Porém, como as etapas mais simples são dominadas primeiro, muitos dos algoritmos simples realizados pelos seres humanos já estão mapeadas em algoritmos computacionais, substituindo as tarefas manuais. São exemplos destas ferramentas os Editores de Máscaras [JAC86][JAC88], Extratores, etc.

Estas ferramentas surgiram na medida em que os projetistas foram percebendo que alguns algoritmos realizados manualmente poderiam ser realizados pelo computador, satisfazendo a dicotomia velocidade e confiabilidade.

Os Geradores de Módulo são ferramentas de síntese automática que abrangem poucos níveis [CAR89]. Um exemplo é o gerador de PLAs [DIL89]. A entrada é uma descrição booleana das equações que se deseja implementar, ou até mesmo a tabela verdade correspondente. Na saída obtém-se uma descrição das máscaras, geralmente independente de tecnologia. Estes geradores surgiram como primeira

alternativa de síntese automática de baixo nível, devido a fatores como conhecimento das técnicas utilizadas pelo projetista, modularidade e regularidade da estrutura.

Os Geradores de PLAs existentes atualmente são capazes de gerar soluções rápidas e com baixo custo (em termos de área), devido aos algoritmos de otimização e "folding" [WES85]. Além disso, é possível obter uma parametrização tanto a nível elétrico como a nível de regras de projeto das máscaras, e, conseqüentemente, uma maior independência tecnológica.

Quando o projetista utiliza um Gerador de PLAs dispensa o uso de uma série de outras ferramentas de CAD, como verificador de regras de projeto, editor de máscaras, simulador elétrico e lógico, etc. As etapas intermediárias entre o nível funcional sob a forma de equações booleanas e o nível geométrico sob a forma de uma descrição das máscaras são percorridas automaticamente pelo gerador de módulos. Deve-se perceber, entretanto, que as soluções geradas por um Gerador de PLAs podem não satisfazer o usuário. Por exemplo, se o projetista dispuser de uma área máxima de 500 micra x 300 micra e o gerador gerou um bloco com uma área de 100 micra x 600 micra. Ou se as entradas do PLA eram no sentido horizontal e as saídas no sentido vertical e o PLA gerado possuir as entradas e saídas no sentido vertical. Mesmo que o gerador satisfizesse todas as combinações possíveis de área, sentido e direção das entradas ainda poderiam existir usuários com novas necessidades, como hierarquia, descrições alternativas, etc.

Porém os geradores produzem módulos bastante particulares e específicos, e que normalmente são configurações mais utilizadas manualmente. Os resultados que

devem ser esperados de um gerador de módulos não devem ser diferentes dos encontrados nas arquiteturas normalmente adotadas.

Como o gerador de módulo produz um módulo ou um conjunto de módulos típicos, cabe ao projetista alterar manualmente os resultados obtidos. Por exemplo, se o projetista precisa de um PLA com características especiais, ele deve modificar ou adaptar a solução obtida as suas necessidades [AYR79b]. Um PLA pode ter várias características topológicas e elétricas impossíveis de serem totalmente parametrizadas.

[GAJ85] refere a uma diferença típica entre duas classes de ferramentas. De um lado ferramentas que auxiliam o projetista na execução de tarefas como: edição de máscaras, extração lógica, verificação de regras de projeto (DRCs), etc. De outro lado as ferramentas de síntese automática. As ferramentas de auxílio ao projeto surgiram com a necessidade do uso de computadores no processo de concepção e projeto de CIs.

O desenho de células dos primeiros circuitos era feito no início sem a utilização de Editores de Máscaras (como acontecia na UFRGS-PGCC até 1984). Basicamente o projetista dispunha de uma folha quadriculada onde realizava o projeto de suas células (note-se a forte característica ascendente de projeto). Após realizar o projeto das células, era necessário digitar as coordenadas dos retângulos. Finalmente as células eram posicionadas em coordenadas calculadas manualmente. O roteamento exigia uma precisão do projetista no cálculo das coordenadas. Em alguns casos, o circuito podia ser exibido em monitores de vídeo em sua configuração final.

A existência de um Editor de Máscaras para um ambiente de desenvolvimento manual pode parecer uma solução definitiva. Porém, se existe uma ferramenta que faz a tradução de uma linguagem do nível de transistores (elétrico) para o nível de máscaras, não é necessário existir um editor de máscaras. Da mesma forma, se a tradução já respeita as regras de projeto, não é necessário verificar a correção destas regras (pois já estão verificadas por construção).

Com o progresso das pesquisas em síntese automática, e a potencialidade progressiva dos computadores, pode-se esperar que boa parte dos projetos de CIs sejam, em breve, realizados automaticamente. A comparação entre dois circuitos que utilizam ferramentas de auxílio ao projeto e ferramentas de síntese automática é interessante. Deve-se perceber que as soluções mais regulares não são necessariamente as melhores. Entretanto, como os geradores de módulos atuais usam estruturas bastante regulares, a comparação com estruturas irregulares feitas manualmente é válida. O avanço das pesquisas nesta área pode levar a soluções irregulares bastante próximas as soluções "full-custom" ou totalmente dedicadas.

No Capítulo IV serão estudadas as principais características de alguns Compiladores de Silício existentes, bem como será definida a arquitetura alvo para geração automática e os motivos de sua escolha.

3. LINGUAGEM DE DESCRIÇÃO

3.1 Introdução

Como visto no capítulo anterior, a linguagem de entrada está em um nível de abstração que se aproxima da linguagem utilizada pelo projetista. Este capítulo apresenta a linguagem de entrada OPCODE. Esta linguagem faz parte do CSAR, que é o compilador de silício proposto neste trabalho.

A linguagem OPCODE é apresentada em comparação a linguagens estruturadas (como PASCAL). É enfatizada a facilidade de descrição de operações em paralelo, e são apresentados alguns exemplos de utilização da linguagem.

3.2 Linguagens de Entrada

A linguagem de entrada de um compilador de silício deve ser o mais abstrata possível, ou seja, o projetista deve se preocupar o mínimo possível com detalhes de implementação, tarefa esta que deve ser feita através de diversas etapas da compilação [AYR79a]. Entretanto, quanto mais abstrata esta linguagem, obviamente mais complexa é a tarefa de chegar a resultados que atendam às especificações de entrada.

Para desenvolver ferramentas que automatizem etapas do processo de compilação é necessário conhecer profundamente as soluções adotadas manualmente pelos projetistas. As linguagens de entrada existentes atualmente são bastante semelhantes as linguagens procedurais normalmente utilizadas em programas de computadores [BAR77] (existem também experiências com linguagens não procedurais [BAR81]). As linguagens de descrição de hardware (HDLs) foram desenvolvidas para permitir a descrição de circuitos, utilizando recursos não disponíveis em outras linguagens de programação e aplicadas a classes de problemas específicos.

Para a linguagem de descrição ser simples, o usuário deve omitir diversas características finais do circuito, que serão consideradas durante a compilação. Este trabalho tem como objetivo o desenvolvimento de um método de geração de circuitos a partir de uma descrição do circuito em um nível abstrato. A descrição do circuito em um nível abstrato pode ser obtida manualmente ou através de outra

ferramenta de síntese de alto nível. As informaçoes apresentadas na descrição serão seguidas na íntegra durante a síntese. As otimizaçöes feitas a partir deste nível de descrição são em relação ao algoritmo de seqüenciamento, compartilhamento de operaçöes mutuamente exclusivas e otimização de variáveis. Estas otimizaçöes, entretanto, são opcionais, e têm suas respostas associadas às restriçöes impostas pelo usuário.

3.3 Linguagem de Entrada OPCODE

Criou-se uma linguagem de entrada bastante simples, contendo as operaçöes que devem ser executadas e a ordem em que devem ser executadas. Esta linguagem de entrada para o Compilador de Silício CSAR é chamada de OPCODE. Uma característica desta linguagem é que ela pode ser interfaceada com outras linguagens mais completas, como ISPS [BAR77]. A inexistência de uma linguagem que tivesse as características que atendessem às necessidades da arquitetura e metodologia de projeto adotadas, influenciaram na decisão de definir uma linguagem própria para descrição de circuitos a serem gerados pelo CSAR.

Para descrição de uma operação em OPCODE é necessário inicialmente informar quando ela é ativada, qual o resultado da operação (transformação de dados) e qual o próximo estado, o qual pode ser uma condição do resultado da operação.

Na fig. 3.1 é apresentado um exemplo de uma descrição de um algoritmo de divisão, encontrado em [JAM85], em uma linguagem semelhante a linguagem de programação PASCAL. Na fig. 3.2 é mostrada a descrição do mesmo algoritmo na linguagem OPCODE. O objetivo desta comparação é de facilitar o entendimento da linguagem proposta, ou seja, não implica que a descrição apresentada seja a melhor descrição de entrada para o compilador.

```
procedure SLOW-DIV(A,B:integer; var R,Q:integer);
{esta procedure divide A por B e retorna o quociente em Q e
o resto em R}

integer C;
begin
   C:=B;
   while C<=A do C:=C*2;
   R:=A; Q:=0;
   while C>B do
      begin
      Q:=Q*2; C=C div 2;
      if R>=C then
         begin
         Q:=Q+1;
         R:=R-C;
         end;
      end;
end;
```

Figura 3.1 - Algoritmo de Divisão em PASCAL-like

```
/* Declaração dos Registradores Utilizados */

Palavra   A 8 e, /* entrada 1    */
          B 8 e, /* entrada 2    */
          C 8 t, /* temporário   */
          R 8 s, /* resto        */
          Q 8 s, /* quociente    */
          1 8 c, /* constante 1 */
          0 8 c; /* constante 0 */

BitSlices 8; /* Número de Bit Slices a ser utilizado */

/* Lógica de Controle e Operacional */

@1 : C <- B ;
#2 : (#3) <- C <= A / #4 ;
#3 : C <- << C / #2 ;
#4 : R <- A / Q <- 0 ( phi 2 ) / (#5) <- C > B / #8 ;
#5 : Q <- << Q / C <- >> C ;
#6 : (#7) <- R >= C / #9 ;
#7 : Q <- Q + 1 / R <- R - C / (#5) <- C > B / #1 ;
#8 : (#5) <- C > B / #1 ;
```

Figura 3.2 - Algoritmo de Divisão em OPCODE.

Na linguagem OPCODE, descreve-se operações que são realizadas em paralelo, separadamente, correspondendo a estados particulares da máquina de estados. Inicialmente são descritos os registradores que serão utilizados na arquitetura. Não foi desenvolvido ainda um algoritmo de otimização de variáveis: os registradores existentes na

descrição são utilizados na realização do circuito. Os registradores são descritos da seguinte forma:

A diretiva "Palavra" indica o início da descrição da lista de registradores. Os registradores são listados (separados por vírgula), contendo as seguintes informações:

Nome do Registrador	Tamanho	Tipo

Onde:

a) O Tamanho do registrador indica o número de bits da palavra.

b) Os registradores podem ser dos seguintes Tipos:

<e> - entrada,
<s> - saída,
<t> - temporário,
<c> - constante,
<x> - reservado.

Os registradores de entrada e saída são automaticamente conectados aos "pads", utilizando "pads" de entrada e saída respectivamente. Um mesmo registrador pode ser descrito como registrador de entrada e registrador de saída. Neste caso é utilizado um "pad" bi-direcional. Os registradores

temporários não são conectados a nenhum "pad", servindo unicamente para armazenar valores temporários, auxiliando a execução de operações. As constantes são registradores que não podem ser modificados, apenas consultados.

O registrador reservado é reservado para futuras expansões da linguagem OPCODE. A diretiva "Bit-Slices" será analisada no Capítulo IV. Após a descrição dos registradores, é necessário descrever as operações que serão realizadas com o objetivo de transformar o conteúdo dos registradores. Descreve-se após a lógica operacional e de controle do circuito a ser gerado, onde cada linha descrita corresponde a um estado da máquina de estados de controle. O estado inicial é indicado pelo símbolo "@".

Uma mesma operação pode estar presente em diferentes estados. No exemplo, a operação / (#5) <- C > B / está presente nos estados #4, #7 e #8. As operações separadas pela barra "/" são executadas em paralelo.

É possível realizar operações sobre um campo dos registradores:

Registrador < bit inicial : bit final >

Ex: R1 < 2:7 > - selecionado o campo do bit 2 ao bit 7.
R2 < 7:2 > - selecionado o campo de bits: 7,0,1 e 2 (considerando que R2 seja um registrador de 8 bits).

Existem três tipos de operações possíveis de serem descritas:

- atribuições

/ registrador destino <- registrador fonte /

Esta operação é utilizada para transferência do conteúdo do registrador fonte ao registrador destino.

- comparações

/ (Verdadeiro) <- comparação / (Falso) /

Esta operação é realizada para comparação e mudança do próximo estado da máquina de estados, conforme o resultado da comparação. Se o resultado da comparação é verdadeiro o próximo estado da máquina de estados é (Verdadeiro), se for falso o próximo estado é (Falso).

- Lógica e Aritmética

/ reg.destino <- reg.fonte 1 (operador) reg.fonte 2 /

A operação lógica ou aritmética é efetuada entre o conteúdo de dois registradores fonte e armazenada no registrador destino.

Existem algumas regras que devem ser respeitadas na descrição:

- Na atribuição:

Os registradores fonte e destino devem ser diferentes. O número de bits transferidos do registrador fonte deve ser igual ao do registrador destino. O conteúdo do registrador fonte considerado é sempre o valor anterior ao da execução de todas operações do estado em consideração.

Ex:

A operação em PASCAL like,

```
{
T = R1;
R1 = R2;
R2 = T;
}
```

pode ser descrita em OPCODE da seguinte forma:

```
#10: R1 <- R2 / R2 <- R1 / #11;
```

- Na comparação:

Se houver mais de uma comparação em paralelo a ordem de precedência é avaliar as operações em seqüência.

Ex1:

```
{
se A > B então estado #3
     senão se C < D então estado #4
          senão estado #5
}
```

```
#10: (#3) <- A > B / (#4) <- C < D / #5;
```

Ex2:

```
{
se A > B ou C < D então estado #3
     senão estado #4
}
```

```
#10: (#3) <- A > B / (#3) <- C < D / #4;
```

```
Ex3:
{
se A > B e C < D então estado #3
     senão estado #4
}

#10: (#4) <- A <= B / (#4) <- C >= D / #3;
```

-Nas operações lógicas e aritméticas:

O registrador fonte 1 pode ser omitido, se a operação lógica for realizada apenas no registrador fonte 2.

Ex:

```
#10: R3 <- ! R2 / R4 <- ^ R5 / #6;
```

Neste caso, as operações "!" e "^" são operações realizadas nos registradores R2 e R5, respectivamente.

Existem alguns conflitos na linguagem que devem ser resolvidos. Por exemplo, caso seja executada uma operação de atribuição em paralelo com uma de comparação, o próximo estado será determinado pela comparação. Logo, para qualquer conjunto de operações, a operação de comparação predomina na determinação do próximo estado.

A linguagem de entrada pode ser classificada como não procedural. Pode-se mostrar uma descrição equivalente na linguagem PROLOG, que pode ser utilizada para simulação do comportamento descrito (fig. 3.3).

A saída do compilador OPCODE fornece os seguintes resultados:

-uma lista dos Módulos Operacionais que devem estar presentes no circuito final. O compilador OPCODE, nesta fase, decide quais operadores podem ser otimizados. A otimização acontece se operações idênticas são realizadas em tempos diferentes. Se, por exemplo, são realizadas em paralelo duas somas, é evidente que serão necessários dois módulos operacionais somadores (o projetista que realizou a descrição deve estar atento a este fato). Se as operações de soma não são realizadas em paralelo (portanto em tempos diferentes), pode-se então utilizar o mesmo módulo operacional somador para implementar as operações de soma.

- uma lista que contém o seqüenciamento de operações, ou seja, que indica qual a seqüencia de operações deve ser realizada e como os resultados de comparações interferem nas próximas operações.

Inicialmente o compilador da linguagem OPCODE foi desenvolvido na linguagem PROLOG. Após foi desenvolvida uma versão em linguagem C, para facilitar a descrição de entrada por parte do usuário.

```
o(1):-atr('C','B'),o(2).
o(2):-cmp('C','<=','A'),o(3).
o(2):-o(4),o(5),o(6).
o(3):-opr('C','C','<<','2'),o(2).
o(4):-atr('R','A').
o(5):-atr('Q','0').
o(6):-cmp('C','>','B'),o(7),o(8),o(9).
o(6):-o(1),o(2).
o(7):-opr('Q','Q','<<','2').
o(8):-opr('C','C','>>','2').
o(9):-cmp('R','>=','C'),o(10),o(11).
o(9):-o(6).
o(10):-opr('Q','Q','+','1').
o(11):-opr('R','R','-','C').
```

onde:

atr - atribuição.
opr - operação lógica.
cmp - comparação.

Figura 3.3 - Slow-Division em PROLOG

4. ESCOLHA DE UMA ARQUITETURA ALVO

4.1 Introdução

Após a definição da linguagem de entrada do CSAR, é necessário especificar a arquitetura a ser utilizada. Este capítulo faz um estudo das arquiteturas utilizadas em outros compiladores de silício e após apresenta um conjunto de alternativas adotadas no CSAR, como número de barramentos variável, utilização de chaves no barramento, sub-partes operativas, etc.

O capítulo 5 é uma continuação deste capítulo, porém enfatiza as estruturas utilizadas na arquitetura e a adaptação as características aqui apresentadas.

4.2 Estudo de Compiladores de Silício

A análise dos compiladores de silício já implementados fornece dados importantes para a construção de novas ferramentas. As versões geralmente encontradas nos compiladores de silício utilizam uma arquitetura fixa ou arquitetura alvo, sendo que estas arquiteturas alvo são divididas em parte operativa e parte de controle, tanto funcional como estruturalmente. O objetivo desta divisão é simplificar o processo de síntese, pois é possível dividir o trabalho em tarefas praticamente paralelas e independentes. Na realidade, porém, em muitos casos a síntese da parte operativa e da parte de controle ainda são mutuamente dependentes.

A parte operativa pode estar dividida em blocos que executam operações simples, como registradores ou unidades lógicas e aritméticas. Os nomes mais usuais para estes blocos são módulos funcionais, módulos operacionais ou unidades de operação e execução. Apesar das diversas nomenclaturas encontradas, o princípio de funcionamento dos blocos da parte operativa é semelhante nas diversas ferramentas de síntese automática. O que muda geralmente é a forma como são utilizados, o tipo de barramento associado e a forma de acesso a este, bem como as atribuições máximas de cada bloco.

A parte de controle é implementada utilizando estruturas bastante regulares, que são mais simples de serem geradas automaticamente. Em alguns casos são feitas apenas adaptações de geradores de módulos em função das necessidades da parte

operativa. Estas adaptaçoes podem implicar em perda de área ou de velocidade no circuito final.

Um estudo dos compiladores de silício que possuem estas características é importante para definir a arquitetura alvo desejada. Para tanto foram selecionados quatro compiladores de silício que possuem características um pouco diferentes entre si. As demais ferramentas analisadas seguem os princípios básicos das apresentadas a seguir, e são apenas citadas como referência ([DUN80], [SZE82], [THO83], [BLA85], [JHO85], [PAR86a], [PAR86b] e [PAR87]).

4.1.1 Datapath [MAR86]

Embora considerado pelos autores como um Silicon Assembler, o programa Datapath utiliza uma série de algoritmos e idéias presentes em outros compiladores de silício. O termo Assembler ou Montador foi inicialmente utilizado para algumas ferramentas com características de Compiladores. Como os montadores possuem uma metodologia de projeto ascendente, o termo compilador é mais adequado e conhecido.

O programa utiliza ICPL, uma linguagem procedural do tipo LISP, para descrição do circuito. A partir da linguagem de descrição o programa gera uma arquitetura utilizando uma biblioteca de células operativas, como registradores, ULAs, barramentos, etc. As células são parametrizadas e independentes de tecnologia. A parte operativa é composta de registradores (RS), unidades de execução (EU) e stack (RS). Estes elementos são ligados através de barramentos globais e

locais. Os barramentos locais são reservados para operaçoes simples, e o barramento global é único.

4.1.2 MacPitts [SOU83]

O programa MacPitts, assim como o Datapath, utiliza uma descrição de entrada semelhante a linguagem LISP. O controle e o paralelismo das microinstruções é expresso pelo usuário.

A arquitetura gerada pelo MacPitts está dividida em organelas, que são uma espécie de sub-partes operativas. As organelas possuem tamanhos variados, e seu posicionamento e roteamento é semelhante ao de circuitos em lógica aleatória. Existe uma biblioteca de células, as quais podem constituir as organelas de forma paramétrica.

4.1.3 SYCO

O SYCO [JER86] é um compilador de silício desenvolvido no IMAG/TIM3-Grenoble para síntese de circuitos do tipo microprocessador, a partir de uma linguagem de entrada algorítmica.

A parte de controle está baseada em uma estrutura composta de múltiplos niveis de interpretação, implementada através de PLAs correspondentes aos diferentes níveis de hierarquia. Com isso, a interpretação de um comando ou operação da linguagem de descrição é efetuada em diversas etapas, que traduzem o comando principal em uma série de sub-comandos de nivel inferior, e assim sucessivamente até a

geração dos microcomandos que atacam a parte operativa (fig.4.1).

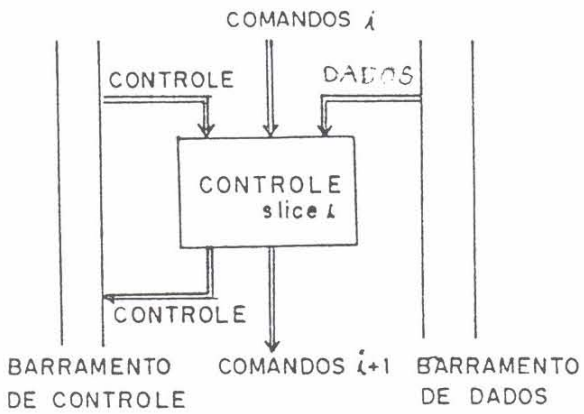


Figura 4.1 Parte de Controle do SYCO

A parte operativa é dividida em "bit-slices" e é composta por registradores, unidades lógicas e aritméticas (ULAs), constantes, etc.

Um conjunto de ULAs e registradores pode ser selecionado ou separado do resto da parte operativa para executar uma determinada operação. Estes blocos são chamados de sub-partes operativas.

As ULAs de uma sub-parte operativa são constituídas de forma que executem apenas as operações que são necessárias. A comunicação entre os elementos da parte operativa é realizada através de dois barramentos, como pode ser visto na fig.4.2.

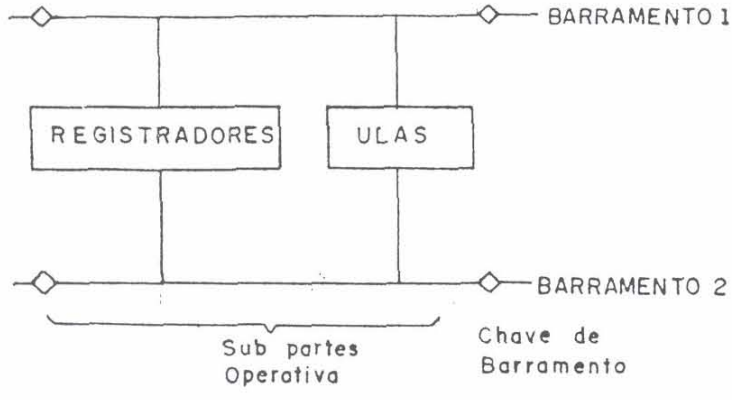


Figura 4.2 Parte Operativa do SYCO

A divisão do barramento em sub-partes operativas é realizada através de chaves de barramento, que são acionadas simultaneamente ou não nos dois barramentos.

Outra ferramenta que segue a linha do SYCO é o DATAPATH.

4.1.4 Arquitetura Assíncrona

Em [HYR86] encontra-se uma arquitetura pouco utilizada em compiladores de silício.

A maior diferença em relação a outras ferramentas está na parte de controle, que é distribuída entre os diversos

blocos da parte operativa, chamados de módulos funcionais (MFs). A topologia final também difere bastante de outras ferramentas. Cada módulo funcional executa uma função específica quando habilitado, podendo executar funções em paralelo com outros módulos funcionais, já que o controle é independente. A forma como os módulos funcionais são ativados é semelhante a uma seqüência de ativação encontrada em uma Rede de Petri. Existe uma marca ou sinal de referência que é distribuído entre os blocos a medida que executam o programa que descreve o comportamento final do circuito (fig.4.3).

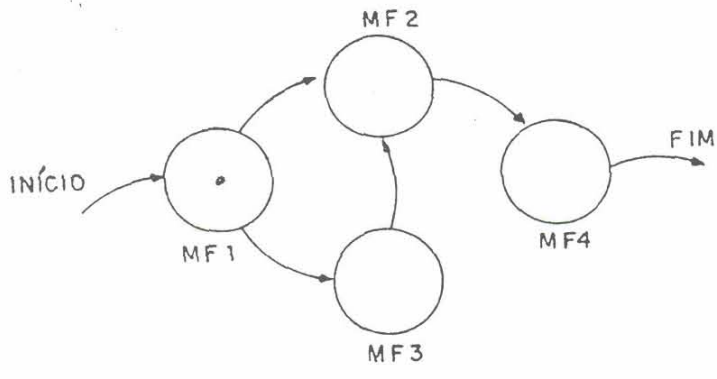


Figura 4.3 Máquina de Controle da Arquitetura Assíncrona

A parte operativa foi dividida fisicamente nas seguintes estruturas:

- módulo de registradores,
- módulo funcional,
- rede de dados e barramento comum.

Na fig.4.4 pode-se ver estas estruturas interligadas.

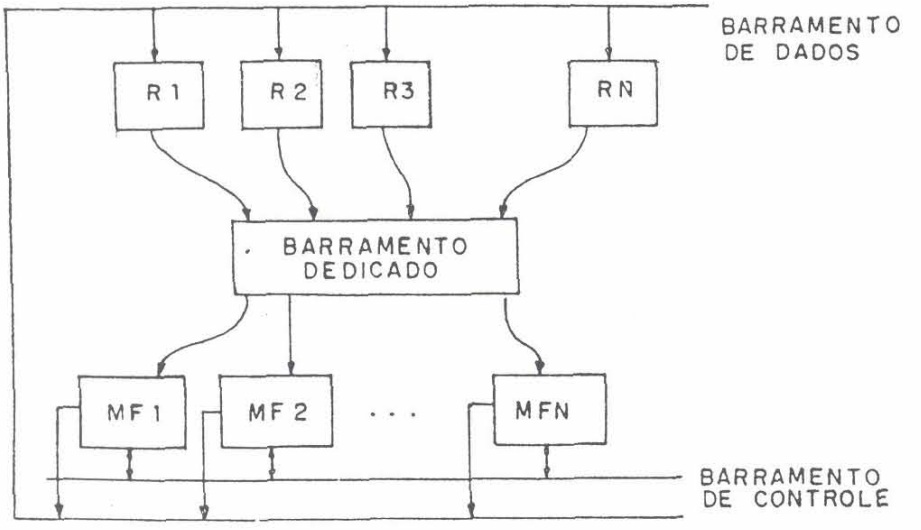


Figura 4.4 Estruturas da Arquitetura Assíncrona

O módulo funcional executa operações aritméticas e lógicas típicas e operações de controle, como teste, chamada de outros módulos, etc. A rede de dados permite transferir os dados dos módulos de registradores até os módulos funcionais. O barramento comum permite a transferência de dados ou controle dos módulos funcionais para o módulo de registradores. A topologia final típica dos circuitos pode ser vista na fig.4.5.

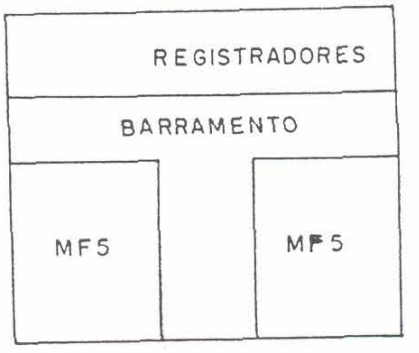


Figura 4.5 Topologia da Arquitetura Assíncrona

Pode-se notar que o módulo de registradores está bastante separado dos demais módulos. Há uma grande área de roteamento entre os módulos funcionais e registradores e nos módulo funcionais entre si.

4.1.5 Comparações

O estudo dos compiladores mostra uma série de características comuns aos programas existentes, como:

- Divisão da arquitetura em parte operativa e parte de controle.

- Parte operativa com registradores e operadores.

- Descrição do paralelismo de microinstruções realizadas pelo usuário.

- Barramentos (exceto o MacPitts).

4.2 Escolha de uma Arquitetura Alvo

A definição de uma arquitetura alvo é importante, pois restringe as aplicações do compilador de silício, facilitando sua implementação. Devem estar bem claras as vantagens e desvantagens da utilização do compilador de silício, bem como o campo de aplicação, para que o usuário (projetista) conheça as limitações da ferramenta.

Procurou-se uma arquitetura que aumentasse o desempenho na execução de operações paralelas, sem comprometer muito a área final do circuito. Portanto, em aumentar o desempenho, entende-se pelo aumento da velocidade de execução das operações sem comprometer a área final. A duplicação de um bloco operativo (uma ULA, por exemplo) pode resolver o problema de paralelismo, porém muitas vezes pode ser desnecessária. Precisa-se estudar novas alternativas, bem como as já existentes, para obter uma solução que aumente o grau de paralelismo das operações sem comprometimento de maior consumo de área.

A arquitetura proposta é do tipo parte operativa e parte de controle, tanto funcionalmente como topologicamente. Quanto à divisão funcional, a parte operativa é responsável pela realização de operações diversas entre os registradores. A parte de controle é responsável pelo sincronismo das operações que são realizadas pela parte operativa. A parte de controle pode tomar decisões a partir de operações realizadas pela parte operativa. Quanto à divisão topológica, normalmente o bloco operativo é constituído de registradores e barramentos de comunicação entre os registradores. Já a parte de controle é formada por um bloco compacto, que pode ser um PLA, uma ROM com um microprograma ou um bloco em lógica aleatória.

A parte de controle se comunica com a parte operativa, enviando sinais de controle e recebendo resultados de operações, utilizados na definição dos passos posteriores.

Os elementos da parte operativa comunicam-se entre si através de um ou mais barramentos. A divisão funcional e topológica em parte operativa e parte de controle facilita o processo de compilação, pois o problema torna-se modular e pode ser atacado em duas frentes praticamente independentes, uma responsável pela geração da parte operativa e outra pela geração da parte de controle.

Algumas arquiteturas que seguem esse modelo são analisadas a seguir:

4.2.1 Arquiteturas Tradicionais

As arquiteturas alvo tradicionais possuem na sua parte operativa registradores e ULAs semelhantes, quando comparadas entre si. As maiores diferenças estão na topologia de barramento. Os barramentos de uma parte operativa podem variar em número, tipo (duplo negado ou único) e topologia em relação aos operadores.

As arquiteturas utilizam geralmente um número fixo de barramentos. A fig. 4.6 mostra arquiteturas com um e dois barramentos.

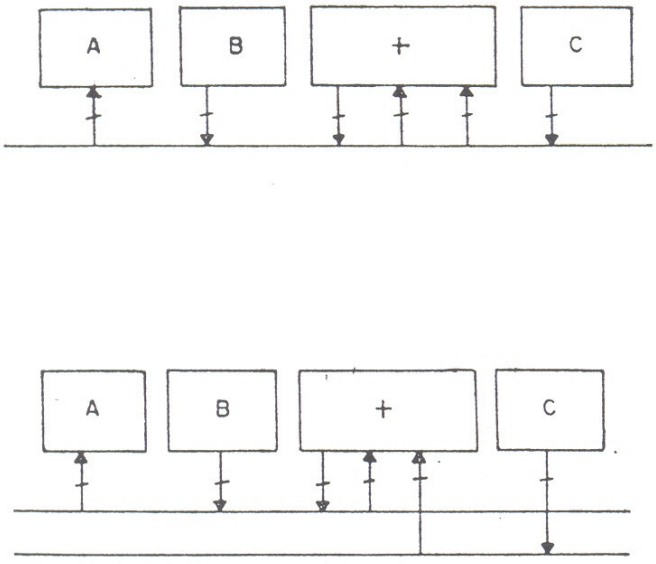


Figura 4.6 Configurações de Barramento Típicas

Havendo apenas um barramento, deve-se prover a parte operativa de registradores de entrada, para armazenar

temporariamente o conteúdo dos registradores envolvidos em operações com mais de um registrador. Numa operação de soma entre dois registradores, por exemplo, o primeiro registrador envia seu conteúdo para o registrador de entrada do somador e após o segundo registrador realiza a mesma operação. Evidentemente os dois registradores não podem utilizar simultaneamente o mesmo barramento.

Geralmente utiliza-se também um registrador na saída da parte operativa, que libera o barramento enquanto é executada a operação. Com a inclusão de mais um barramento aumentamos a velocidade de execução das operações, através da carga paralela dos registradores envolvidos na operação.

Neste caso é possível excluir os registradores de entrada da ULA. O resultado é enviado ao registrador de destino da operação através de um dos barramentos. Com um terceiro barramento pode-se liberar os demais para carga dos registradores de entrada enquanto é armazenado o resultado obtido.

Portanto, assim como na máquina Von-Neuman, o "gargalo" de um processador são as operações de E/S, que utilizam o barramento de dados, o qual deve ser coerentemente projetado para a aplicação desejada, sob pena de comprometimento do desempenho do circuito.

Uma quarta solução, encontrada em [PAR86], consiste em dividir a parte operativa em vários níveis hierárquicos, de forma a executar seqüencialmente as diversas operações em paralelo, numa espécie de pipeline a nível de utilização

da ULA e registradores, procurando executar uma operação por ciclo de máquina (fig. 4.7).

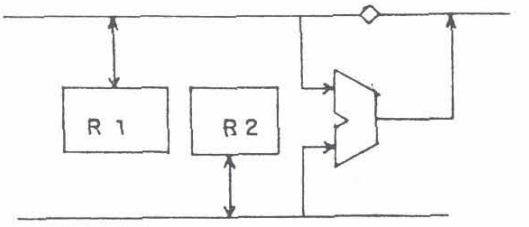


Figura 4.7 Parte operativa compartilhada

Para aumentar a realização de operações paralelas é necessário, como primeira medida aumentar o número de ULAs. Entretanto, devido ao tamanho destas, isto pode ser custoso em área. Uma descrição que possua três somas em paralelo exigiria três ULAs, embora para as demais operações (subtração, deslocamento, etc.) seja suficiente apenas uma ULA.

A melhor solução é, portanto, dividir a ULA em pequenos operadores que realizem uma única operação. Assim, existe um operador para soma, outro para subtração, etc. Com isso, para uma classe de problemas a resolver, consegue-se reduzir a área da parte operativa sem diminuir

o paralelismo na execução das operações previstas para o processador em desenvolvimento.

4.2.2 Número de Barramentos Variável

Quanto ao número de barramentos, observou-se que um número fixo de barramentos comprometia bastante o objetivo final do trabalho, que é permitir um melhor desempenho na execução de operações paralelas.

O ideal, neste caso, seria criar um algoritmo que interpretasse a descrição de entrada do circuito e analisasse a melhor solução, apresentando como resultado o número de barramentos ideal. Procurou-se então encontrar uma maneira de tornar o número de barramentos mais flexível e dependente da descrição de entrada. Para atingir este objetivo (número de barramentos flexível) é necessário encontrar uma solução algorítmica e uma solução topológica para o problema. A solução algorítmica é composta por um algoritmo de posicionamento e um algoritmo de cáculo do número de barramentos necessários. O posicionamento, como será visto, é realizado seguindo diversos critérios especificados pelo usuário.

4.2.3 Bit-Slice

A idéia é posicionar todos os elementos do bloco operativo numa estrutura "bit-slice" (como normalmente é feito em outros compiladores de silício), porém sem reservar área para passagem de barramentos (todas as células desenhadas com a altura mínima possível). A comunicação entre os diversos elementos da parte operativa é feita por

um ou mais barramentos, que formam um canal de roteamento entre os planos P e N (fig. 4.8).

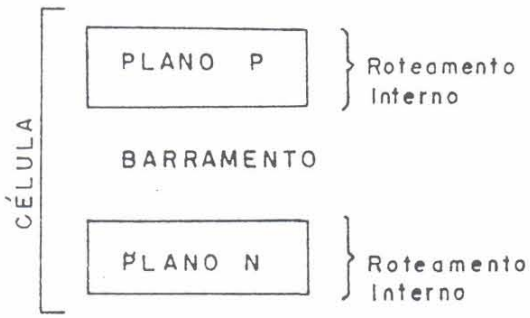


Figura 4.8 Configuração de Barramentos no CSAR

Em outras palavras, o "bit-slice" forma uma banda (como as encontradas em circuitos semi-dedicados) e a comunicação entre os diversos elementos do "bit-slice" é feito através de um canal de roteamento.

4.2.4 Chaves no Barramento

As chaves de barramento são uma poderosa solução para aumentar o paralelismo de operações em qualquer arquitetura.

Como exemplo, supondo que as seguintes operaçoes devam ser executadas em um único ciclo (composto de duas fases):
1: R1 <- R2 + R3 / R4 <- R5 + R6

Para existir um barramento dedicado é necessário que neste barramento sempre estejam disponíveis todas as saídas dos registradores, de modo a realizar qualquer conjunto de operaçöes em paralelo em um único ciclo de relógio.

Para o exemplo, se for utilizado um barramento para cada registrador consultado, são necessários quatro barramentos, como mostra a fig. 4.9.

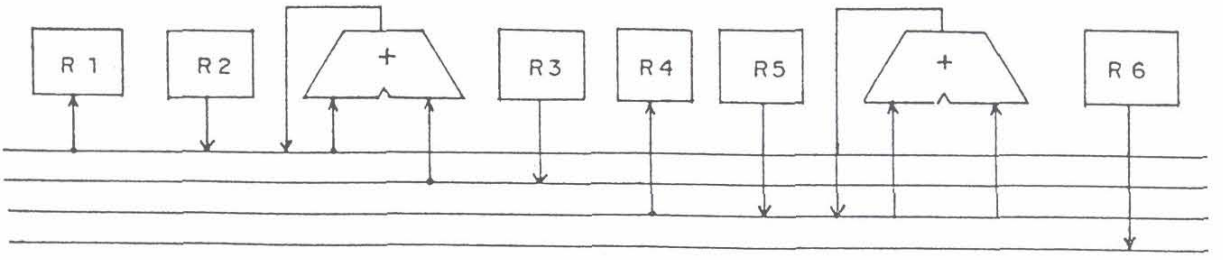


Figura 4.9 Solução com Barramento Dedicado

Adotando-se a tradicional solução de dividir o barramento com chaves, chega-se à dois barramentos (fig. 4.10) ou um único barramento (fig. 4.11).

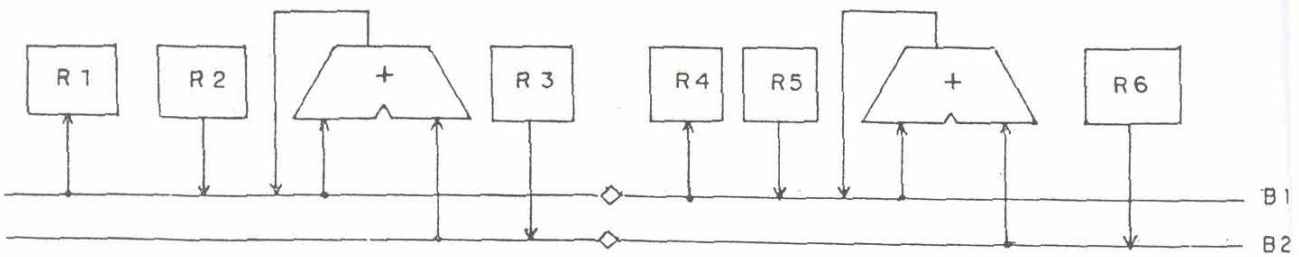


Figura 4.10 Solução com Chaves no Barramento

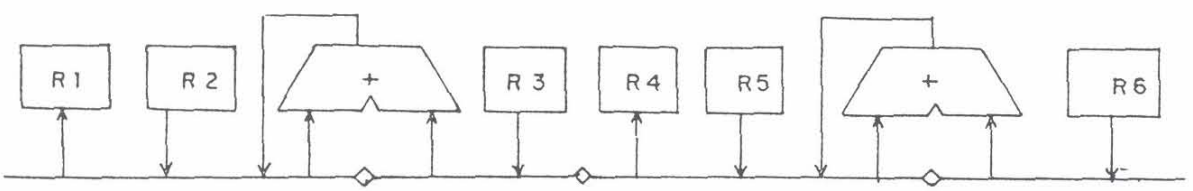


Figura 4.11 Outra solução com Chaves no Barramento

Esta solução, apesar de aumentar a lógica de controle, devido às chaves de barramento, apresenta duas vantagens claras:

- Menor área final.
- Menor capacitância no barramento.

A menor capacitância no barramento só é válida quando o barramento é dividido através das chaves.

4.2.5 Módulos Operacionais (MOs)

Os Módulos Operacionais (MOs) realizam funções normalmente encontradas nas Unidades Lógicas e Aritméticas (ULAs). Logo, o Módulo Operacional é responsável pelas operações lógicas e pelas operações aritméticas que ocorrem entre os registradores.

Existem algumas características bastante particulares nos MOs que diferem das ULAs tradicionais:

- Os MOs não implementam necessariamente as mesmas funções, ou seja, os MOs não são sempre idênticos. Pode-se ter um MO específico para realizar operações de deslocamento, ou um MO específico para operações de soma e subtração.

- Vários MOs podem ser agrupados em um único MO (neste caso o MO torna-se mais semelhante à uma ULA). A fig.4.12 mostra um exemplo de agrupamento de dois MOs, para uma determinada arquitetura que implementa as operações:

a := b + c; a := b . c;

O agrupamento de MOs busca uma maior economia na área final do circuito, um número menor de chaves de entrada e saída. Por outro lado, a capacitância de entrada do MO final aumenta.

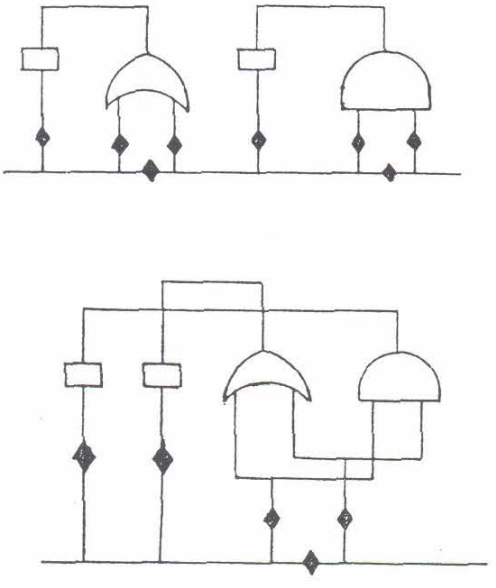


Figura 4.12 Agrupamento de Módulos Operacionais

4.2.6 Chaves Internas nos MOs

Para entender esta aplicação inserida no Compilador de Silício proposto, basta analisar o seguinte exemplo.

Supondo que seja necessário realizar a seguinte operação:

```
a <- b + c;
```

Deve então ser criada uma estrutura especifica para

somar dois registradores e atribuir o resultado a um terceiro registrador. O exemplo, apesar de simples, conduz a uma solução bastante interessante. Com o objetivo de ocupar a menor área possível, adota-se a solução da fig. 4.13, que utiliza apenas um barramento.

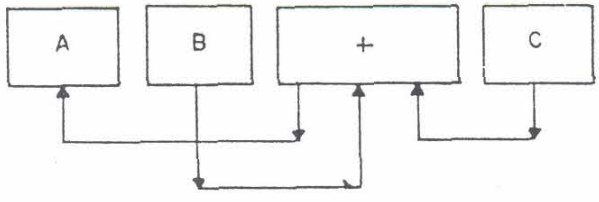


Figura 4.13 Solução com Um Barramento

Entretanto, como a transferência do conteúdo dos registradores "b" e "c" não pode ser realizada em um único ciclo, é necessário no mínimo dois ciclos de relógio. No primeiro ciclo de relógio, o conteúdo do registrador "b" é armazenado pelo registrador de entrada do somador. Após este conteúdo é somado ao registrador "c" e finalmente o resultado da soma é transferido para o registrador "a". Para aumentar a velocidade de execução da operação e diminuir a área final do

circuito retirando o registrador de entrada do somador, a solução sugerida pela bibliografia é utilizar dois barramentos, como na fig. 4.14.

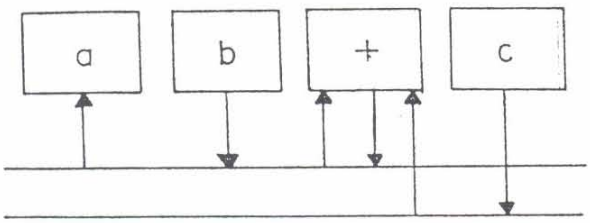


Figura 4.14 Solução com Dois Barramentos

Sem dúvida a solução aumenta a velocidade de execução, pois necessita de apenas um ciclo para a soma dos registradores "b" e "c". Uma segunda solução para o problema é mostrada na fig. 4.15. Nesta solução simplesmente é inserida uma chave no barramento de forma estratégica.

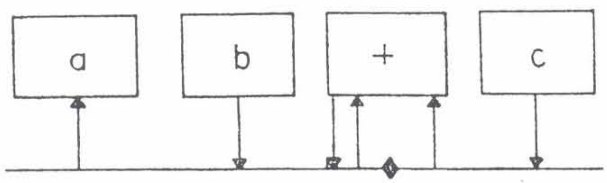


Figura 4.15 Solução Alternativa com Chave no Barramento

A posição da chave inserida é entre as entradas do MO. A vantagem desta solução, em comparação com a solução de utilizar um outro barramento, está na área final do circuito. Deve-se considerar que a lógica de controle torna-se mais complexa com o aumento do número de chaves no barramento.

Um diagrama dos MOs com chaves internas pode ser visto na fig. 4.16.

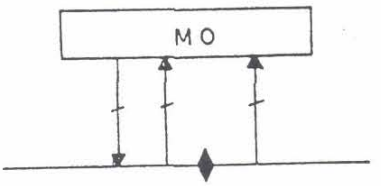


Figura 4.16 Diagrama dos MOs com Chaves Internas

A utilização de chaves no barramento como feita no exemplo anterior, conduz a uma nova metodologia para escolha de chaves em barramento. As chaves de barramento normalmente são responsáveis pela divisão da parte operativa em um conjunto de sub-partes operativas, que realizam operações em paralelo. Contudo as sub-partes operativas apresentam um conjunto bem definido de registradores e ULAs.

Para a solução proposta, pode-se dizer que duas sub-

partes operativas passam a compartilhar de uma mesma ULA ou MO.

4.2.7 Saídas Multiplexadas

Uma facilidade inserida no CSAR é a possibilidade de multiplexação das saídas dos registradores, quando necessário. A multiplexação permite que a mesma saída de um registrador seja compartilhada por mais de um barramento. Como exemplo, caso seja necessário implementar um circuito com o seguinte conjunto de operações:

a <- b + c; b <- a + c; c <- a + b;

Se o conjunto de operações for implementado utilizando dois barramentos, chega-se a solução da fig. 4.17.

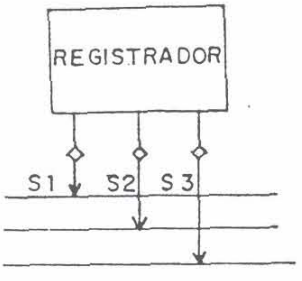


Figura 4.17 Multiplexação da Saída de Registradores

Nesta solução foi necessário criar um multiplexador para as saídas do registrador "c", para que estas saídas possam

estar simultaneamente presentes em dois barramentos.

4.2.8 Sub-Partes Operativas

A utilização de sub-partes operativas implica na divisão de todos os barramentos, como mostra a fig. 4.18.

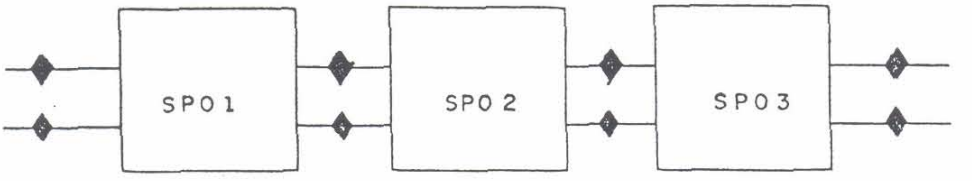


Figura 4.18 Divisão do Barramento em Sub-Partes Operativas

Pode-se afirmar que não existe uma prova coerente de que a adoção de sub-partes operativas conduza a solução de melhor desempenho, quanto à área e velocidade final do circuito.

Como exemplo, caso seja necessário implementar um circuito com o seguinte conjunto de operações:

a <- b + c / b <- a + c

A primeira tentativa de resolver o problema utilizando dois barramentos conduz a solução da fig.4.19.

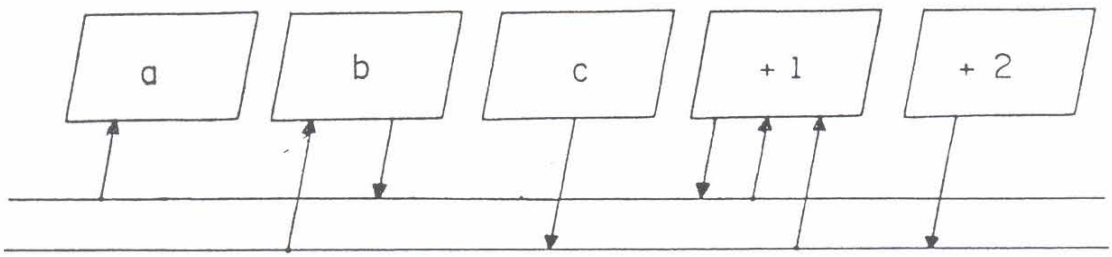


Figura 4.19 Primeira Tentativa com Dois Barramentos

Inicialmente foi implementada a operação: a <- b + c , utilizando o somador 1. Quanto à segunda operação: b <- a + c, não houve como implementá-la, pois o registrador "b" entra em conflito com o registrador "a" quando forem utilizar o barramento. Alterando o posicionamento dos registradores e MOs pode-se chegar a solução da fig. 4.20. Nesta solução é inserida uma chave no segundo barramento, que evita o conflito dos registradores "a" e "b".

Note-se que a utilização de chaves no barramento não é necessariamente realizada em todos os barramentos. Também é importante notar que existem duas sub-partes operativas. Em 4.2.5 as sub-partes operativas compartilhavam um MO. Neste caso as sub-partes operativas compartilham o registrador "c".

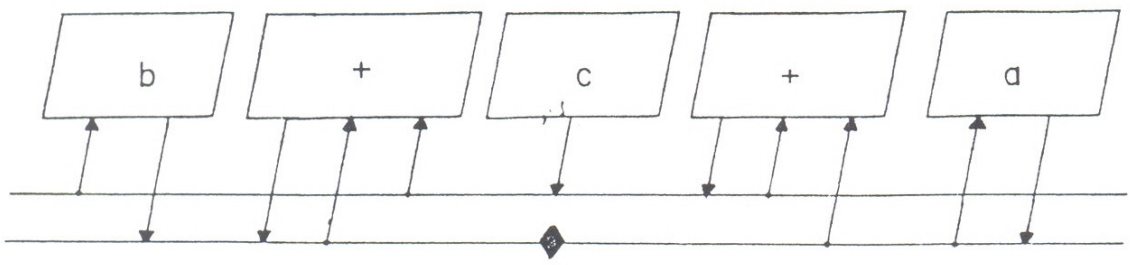


Figura 4.20 Segunda Tentativa com Dois Barramentos

A divisão de sub-partes operativas no CSAR passa a ser mais funcional sem ser realmente topológica.

4.2.9 Avaliações

As características principais da arquitetura alvo proposta podem ser resumidas nos seguintes pontos:

- Arquitetura do Tipo Parte de Controle e Parte Operativa.

- Parte Operativa dividida em "bit-slices".

- O número de barramentos depende da complexidade da descrição de entrada do circuito.

- Módulos Operacionais realizam as operações.

- Os MOs podem possuir chaves internas para dividir o barramento.

- As chaves nos barramento não possuem nenhuma simetria ou regra fixa. Simplesmente são posicionadas em função das necessidades.

Com isso, pode-se chegar a algumas conclusões sobre a arquitetura alvo adotada:

-complexidade do barramento dependente da complexidade da descrição.

-o número de operaçoes em paralelo pode ser aumentado mantendo-se a execução destas em um ciclo.

-utilização de ferramentas já desenvolvidas e conhecidas, como posicionadores e roteadores.

-dispensa registradores de entrada nos elementos operativos (e como será visto mais adiante, também são eliminados os registradores de saída, através da utilização de lógica dinâmica).

-adapta-se à síntese de circuitos com pouco paralelismo de operaçoes assim como à síntese de circuitos com alto grau de paralelismo.

-torna inviável a utilização de barramentos duplos (ou seja, barramentos complementares), devido ao aumento crítico da altura final do canal de roteamento.

5. PARTE DE CONTROLE E PARTE OPERATIVA

5.1 Introducão

Este capítulo apresenta as estruturas que fazem parte da arquitetura alvo escolhida. As estruturas principais da arquitetura são: módulos operativos (ou operadores) e registradores. Um circuito que implementa um algoritmo constitui-se de uma série de transformações dos valores dos registradores através dos operadores, até atingir um estado desejado de valores dos registradores. A ordem em que as operações são realizadas é determinada pelas estruturas de controle.

Além da definição das estruturas é apresentado o algoritmo de posicionamento, roteamento e cálculo do número de barramentos do CSAR.

O capítulo encerra com uma análise de irregularidades das estruturas do CSAR.

5.2 Arquitetura da Parte Operativa

É apresentado inicialmente o diagrama lógico dos elementos que compõem a parte operativa, para depois ser mostrado como é implementada a comunicação entre os elementos e os sinais de controle.

5.1.1 Barramentos

O barramento é implementado como um canal de roteamento que interliga os diversos módulos operativos. Devido a complexidade deste canal, é muito onerosa a uitilização de barramento complementar, o que permitiria conectar facilmente registradores do tipo memória. Optou-se descartar a utilização de registradores do tipo memória.

Escolheu-se, portanto, aumentar a complexidade dos registradores com o objetivo de reduzir o número de barramentos. Foi adotado também a utilização de pré-carga no barramento, pelos seguintes motivos:

- diminuição da capacitância de barramento [CAR88].

- diminuição do tamanho das células, com a eliminação dos transistores do tipo p, substituídos por um único transistor de pré-carga. Esta solução visa uma compensação da área exigida pelo barramento.

- diminuição do tamanho dos operadores, eliminando o registrador de saída, devido à lógica de pré-carga na saída dos operadores.

Como conseqüência, tem-se uma perda em desempenho, pois a lógica de pré-carga exige um relógio com o dobro do número de fases.

5.1.2 Registradores

Como os registradores não estão todos envolvidos em operações a cada ciclo de relógio, os registradores devem ser estáticos, ou seja, devem guardar a informação por tempo indeterminado. Pela lógica envolvida na execução das operações, não há necessidade de serem do tipo mestre-escravo, pois utilizam os registradores de saida dos elementos operativos para suprir esta função.

Como o número de sinais no canal de roteamento já é grande, a utilização de registradores com apenas seis transistores exigiria a duplicação (negação) de alguns sinais, comprometendo demais a área final. Portanto adotou-se uma lógica estática de oito transistores, como vista na fig. 5.1.

A conexão dos registradores ao barramento pode ser feita através de uma ou mais chaves como na fig. 5.2 (o aumento do número de chaves e os motivos deste aumento serão vistos posteriormente).

Pode-se ver que a lógica funciona utilizando duas fases de relógio. Em PHI1 (fase 1) é sempre habilitada a realimentação (responsável pelo "refresh") e leitura do registrador (caso habilitado pelo sinal RRn, que indica habilitação de leitura, e é ativado pela parte de

controle). Em PHI2 (fase 2) é feita a escrita do registrador (caso habilitado por WRn, que indica habilitação de escrita. Pode-se ver também que a saída do registrador pode estar em curto-circuito com a entrada, já que a sincronização garante que as chaves nunca são ativadas simultaneamente.

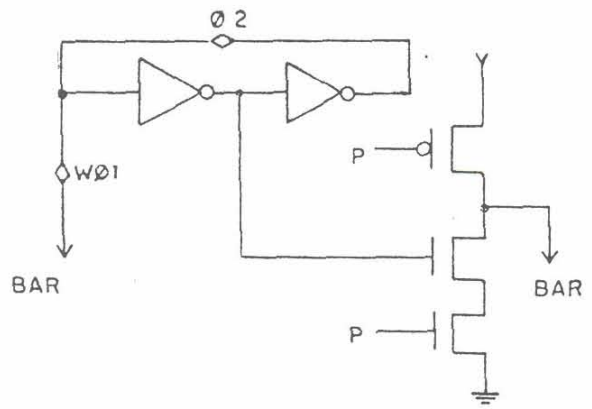


Figura 5.1 Lógica Estática dos Registradores

Para conectar as saídas dos registradores ao barramento (bem como as saídas de outros módulos) utiliza-se uma lógica dinâmica com pré-carga. A lógica de pré-carga de barramento funciona da seguinte forma: inicialmente é feita a pré-carga do barramento através da ativação do sinal P (lógica negada) da fig. 5.3.

O barramento é carregado então pelo transistor de pré-carga (transistor pmos). Após a pré-carga (sinal P = "1") o barramento pode ser descarregado ou não, conforme o conteúdo do registrador. A lógica de pré-carga apresenta as seguintes vantagens:

- simplifica o número de sinais de controle,
- diminui a carga no barramento, pois diminui o número de transistores,
- possui área menor em comparação à chave CMOS, pois o transistor de pré-carga é comum para todo o barramento,
- maior velocidade nas operaçoes.

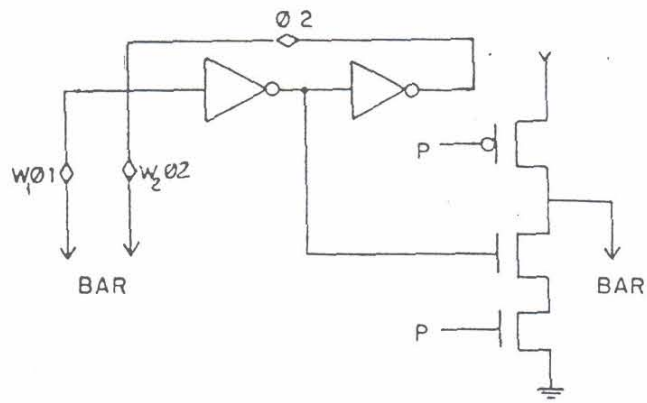


Figura 5.2 Conexão Multiplexada dos Registradores ao Barramento

O sinal RR habilita a leitura do registrador. Se a saída do registrador é "1" lógico o transistor nmos não é ativado e a saída se mantém no nível de pré-carga. Se a saída do registrador é "0" lógico o transistor nmos conduz (note que o sinal RR é ativado pela saída negada do registrador) e o barramento é descarregado.

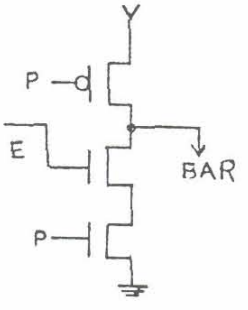


Figura 5.3 Lógica de Pré-Carga do Barramento

5.1.3 Operadores

Os operadores são responsáveis pela transformação dos dados recebidos pelos registradores e pela devolução do resultado para outro registrador, realizando assim uma operação. Na arquitetura proposta procurou-se adotar operadores que utilizassem apenas um ciclo de relógio na

execução de uma operação. Os operadores são: somadores/subtratores, deslocadores lógicos, portas lógicas e comparadores.

Note-se que, pela característica do barramento, não são necessários operadores para atribuição direta, que estão implícitos no canal de roteamento, ou seja, efetua-se apenas uma transferência entre registradores que estão conectados a um barramento. Igualmente não é necessário registradores de entrada para os operadores. Existe apenas um conjunto de chaves que controla a entrada dos registradores e que automaticamente permite a multiplexação entre eles. A lógica de pré-carga na saída evita a necessidade de um registrador de saída nos operadores, como nos casos em que o registrador que recebe o resultado da operação está envolvido no cálculo.

A estrutura do CSAR, permite que o usuário crie seus próprios Módulos Operacionais. Entretanto alguns MOs básicos foram desenvolvidos, e fazem parte de uma biblioteca de células parametrizável.

Os operadores implementados são:

-soma/subtração,

-comparação (>,<,>=,<=,!=,=),

-lógica (and, or ,xor ,inversor e deslocamentos).

Para possibilitar a realização destas operações são

necessários sete módulos operacionais distintos, compostos por sete células básicas:

- MO1 : SOMADOR / SUBTRATOR.
- MO2 : COMPARADOR.
- MO3 : AND.
- MO4 : OR.
- MO5 : XOR.
- MO6 : INVERSOR.
- MO7 : DESLOCAMENTO.

Os MOs 6 e 7 possuem uma entrada e uma saída, em sua configuração mínima. Os MOs de 1 a 5 possuem duas entradas e uma saída em sua configuração mínima. Os registradores também podem ser considerados Módulos Operacionais, cuja função básica é armazenar o resultado das operações.

5.1.3.1 MO1: Somador / Subtrator

A utilização de somadores que se adaptem às características topológicas e de desempenho já apresentadas deve respeitar algumas regras básicas:

- O somador deve ser dividido em bit-slices.

- As entradas devem possuir facilidade de acesso aos barramentos, assim como os sinais de saída ou resultado da soma.

- As saídas e entradas de carry (vai um) devem ser implementadas em sentido perpendicular aos barramentos.

- O atraso de propagação do sinal de carry deve ser o mínimo possível, pois este atraso limita a velocidade final de operação. A adoção de soluções do tipo "carry-look-ahead" deve prever irregularidades na topologia final, como é visto no item 5.5.

Como exemplo de um circuito adequado a estas regras pode-se analisar a solução com "n-bit ripple carry", mostrada na fig. 5.4.

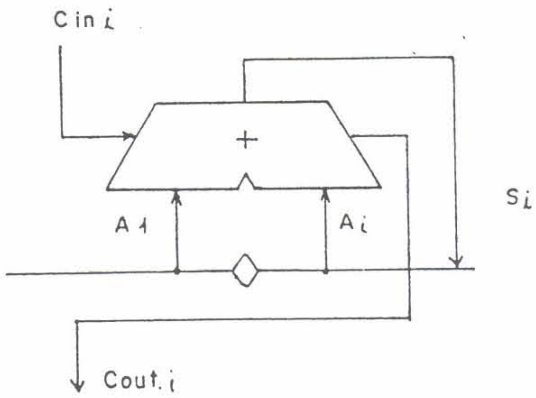


Figura 5.4 Somador "n-bit ripple carry"

Pode-se ver pela figura que apesar da lógica ser bastante simples, existe uma série de irregularidades, principalmente na primeira e na última célula. Para que o problema das irregularidades seja resolvido, neste caso, é necessário que exista uma biblioteca com no mínimo três células: célula inicial, célula intermediária e célula final.

A montagem da estrutura do somador é efetuada da seguinte forma:

Somador = célula inicial +
n * célula intermediária +
célula final.

O diagrama elétrico de uma célula do tipo "ripple carry" pode ser visto na fig. 5.5.

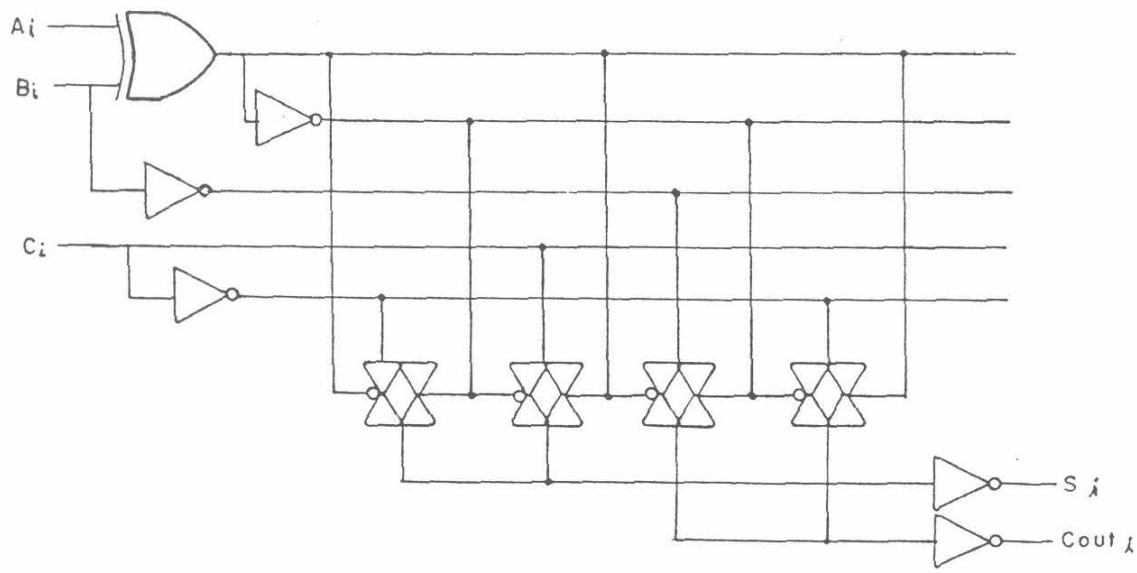


Figura 5.5 Diagrama Elétrico do Somador

Em [WES85] encontra-se uma série de diagramas lógicos para geração dos sinais Si e Couti. Como a arquitetura do CSAR exige que os Módulos Operacionais executem uma operação em um ciclo de duas fases, o somador deve utilizar uma configuração compatível. O somador "n-bit ripple carry" realiza uma operação em apenas um ciclo. Entretanto o atraso acarretado pela lógica de carry pode limitar a freqüência de operação da máquina de estados do circuito gerado.

5.1.3.2 MO2 : Comparador

O circuito comparador utiliza um somador em cada bit-slice. As saídas dos somadores são entradas de uma porta NOR, que realiza a comparação (fig. 5.6).

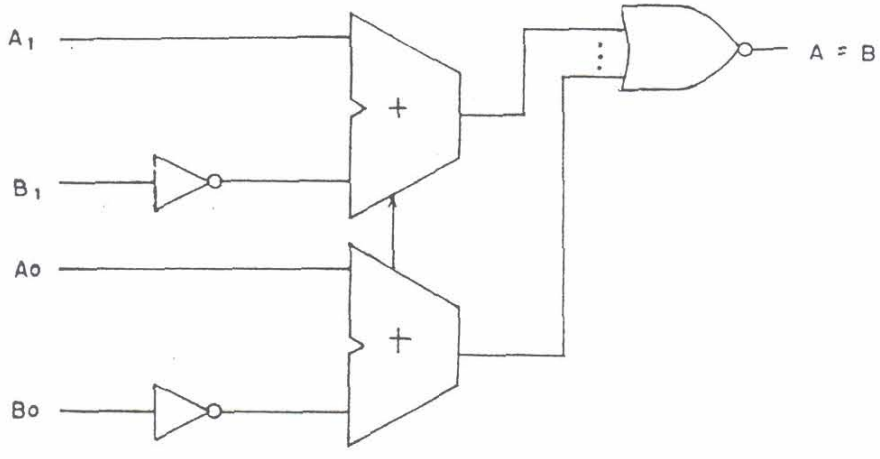


Figura 5.6 Diagrama Elétrico do Comparador

O comparador possui uma característica interessante: este Módulo Operacional apresenta duas entradas conectadas ao barramento, porém a saída ou resultado da comparação não é conectada a nenhum barramento. A saída do comparador (sinais B>A e A=B) é dirigida à parte de controle, e fará parte do cálculo do próximo estado da máquina de estados.

5.1.3.3 MO3-6: And, Or, Xor, Inversor

As portas lógicas And, Or e Xor possuem duas entradas e uma saída ligadas aos barramentos. Não possuem nenhuma particularidade especial, a não ser que devem respeitar as características de conexão ao barramento da arquitetura alvo.

Na fig. 5.7 podem ser vistos os diagramas lógicos e elétricos destas portas lógicas.

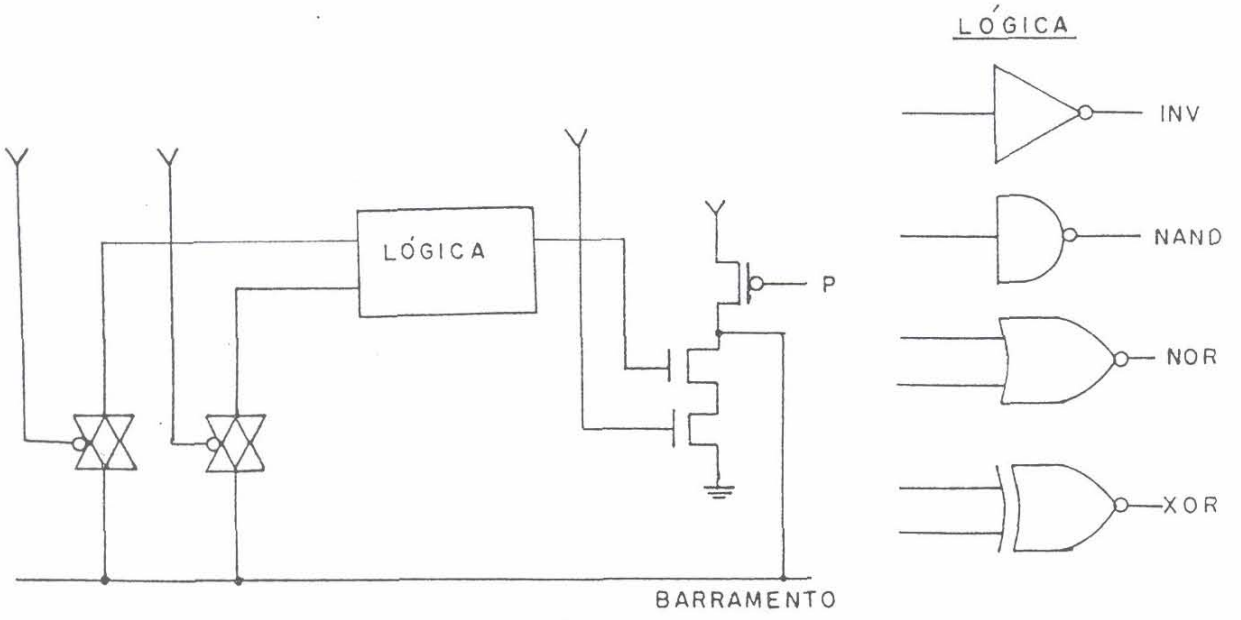


Figura 5.7 Diagramas das Portas Lógicas

5.1.3.4 MO7 : Deslocamento

O módulo operacional de deslocamento realiza apenas uma operação de desvio dos sinais de um bit-slice para outro bit-slice.

O diagrama lógico do circuito de deslocamento lógico é mostrado na fig. 5.8.

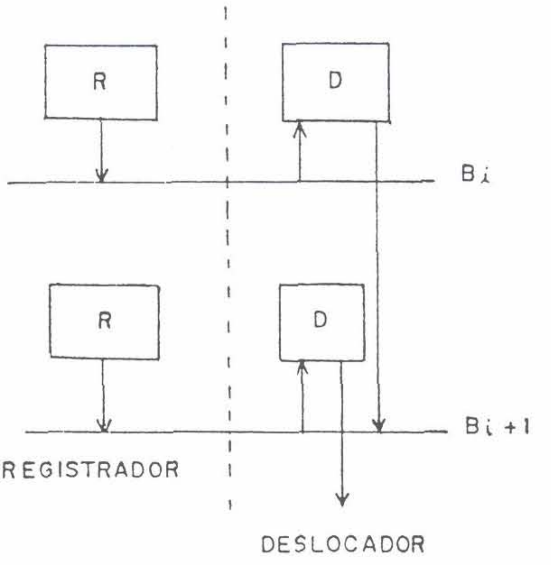


Figura 5.8 Diagramas do Deslocador Lógico

5.2 Posicionamento

Após a compilação dos dados de entrada, o programa deve definir quais módulos operacionais comporão o circuito, bem como suas funções associadas a cada estado e suas interligações. Os módulos operacionais são posicionados de forma a obter-se uma solução em termos de área e velocidade adequadas à especificação. O resultado do posicionamento influencia de forma direta no número final de barramentos a ser utilizado, bem como no número de chaves de barramento.

O problema de posicionamento está presente em diversas metodologias de projeto VLSI, e, portanto, vem sendo largamente estudado e pesquisado (assim como o problema de roteamento).

Tanto em circuitos semi-dedicados, como em pré-difundidos, o posicionamento é uma etapa importante e seus resultados influenciam muito o desempenho do circuito final. Na síntese da arquitetura proposta, o problema é simplificado por envolver apenas uma dimensão, ou seja, o posicionamento é feito em uma única fatia de "bit-slice".

Em [BRY86] encontramos um excelente resumo das diversas técnicas de posicionamento existentes, assim como em [FLA87]. Estas técnicas se dividem basicamente em dois grandes grupos:

- posicionadores construtivos.

- posicionadores iterativos.

No primeiro grupo, os elementos estão inicialmente esperando o lugar onde serão alocados e este posicionamento depende de vários critérios e funções peso, que vão se alterando na medida em que novos elementos são alocados. Quando todos elementos estiverem alocados, o posicionador chegou ao resultado.

Já no segundo grupo, os elementos já possuem um lugar inicial, portanto já estão alocados. São então realizadas diversas iterações, de forma a atender uma função custo, que pode ser, por exemplo, a menor soma de comprimentos de ligações entre os elementos. As técnicas construtivas existente são bastante complexas para problemas bidimensionais.

Resolveu-se adotar uma solução mais simplificada para

obter-se um posicionamento construtivo inicial. Esta solução está baseada na técnica de crescimento de grupos, pois adapta-se bem para uma dimensão apenas e é relativamente simples. A técnica de crescimento de grupos consiste em selecionar um elemento raíz ou semente que é posicionado inicialmente. Os demais elementos são seqüencialmente selecionados e posicionados em relação aos componentes já posicionados, seguindo uma função peso ou custo. Para o Compilador de Silício proposto, esta técnica apresenta a desvantagem de não deixar muitas alternativas ao usuário de interferir no resultado.

A técnica de crescimento de grupos segue normalmente regras fixas, que embora sejam muitas vezes uma boa solução, nem sempre buscam um equilíbrio de desempenho, tanto em área final como em velocidade final. Pensando em diversificar as soluçoes obtidas pelo posicionador, de modo a possibilitar ao usuário a escolha de uma opção que resolva melhor seu problema, bem como possibilitar um refinamento da solução obtida construtivamente, resolveu-se adicionar um posicionador iterativo, com tempo de execução especificado pelo usuário. Com isso o posicionamento tem um tempo mínimo de execução que pode ser aumentado, conforme as disponibilidades de tempo, resultando em soluçöes mais otimizadas.

O posicionador iterativo parte de uma solução inicial (vinda do posicionador construtivo) e gera três soluçöes, que apresentam vantagens em termos de àrea ou velocidade.

5.2.1 Posicionador Construtivo

A entrada de dados deste posicionador é por intermédio de um arquivo que contém a especificação do número e do tipo de módulos funcionais a serem posicionados e seu relacionamento. O relacionamento é feito a nível de blocos e não de sinais, tornando o posicionamento mais rápido. Como exemplo, suponha uma descrição inicial que contenha apenas duas operações, como:

1) R1 <- R2 + R3 / R2 <- R3 and R2

São necessários para implementação física destas operações apenas cinco módulos operacionais, a saber:

MO1) R1; (registrador 1)
MO2) R2; (registrador 2)
MO3) R3; (registrador 3)
MO4) +; (somador)
MO5) and; (porta E)

A entrada de dados pode ser:

Tempo máximo: 200s
Número de MOs: 5
Número de Relacionamentos: 5
Relac1: 2,4 (R2 com +)
Relac2: 3,4 (R3 com +)
Relac3: 1,4 (R1 com +)
Relac4: 2,5 (R2 com and)
Relac5: 3,5 (R3 com and)

Os relacionamentos envolvem qualquer conexão de E/S. Os

sinais envolvidos não são considerados, como por exemplo, entrada 1 do somador ligada à saída do registrador R2. Considera-se apenas que existe um relacionamento entre o registrador R2 e o somador.

Pode-se ver que embora existam dois sinais envolvidos entre R2 e a porta E (saída da porta E ligada a entrada de R2, saída de R2 ligada a entrada da porta E), apenas um relacionamento é especificado. Procura-se, deste modo, diminuir o número de variáveis envolvidas no posicionamento, que pouco influenciarão no resultado. Esta atitude é adequada quando considera-se o maior comprimento de todas as ligações como o mais importante.

Uma vez lido o arquivo de entrada inicia-se a execução do algoritmo construtivo, que cria uma tabela com todos os elementos, sua posição e o número de conexões. O módulo operativo que possuir o maior número de conexões é o elemento semente ou raiz. Este elemento ocupa então um lugar na tabela, que poderá ser alterado, conforme necessário. A escolha do próximo elemento a posicionar é feita através de uma pesquisa do elemento com mais conexoes com os elementos já posicionados. Este elemento tem sua posição escolhida pelo critério de menor comprimento máximo de suas conexões ou pela soma do comprimento de todas as conexões do elemento, conforme peso atribuído pelo usuário. A definição do peso do menor comprimento máximo das conexões ou da soma do comprimento de todas as conexões do elemento é a única influência que pode ser feita pelo usuário no posicionamento construtivo. Os demais elementos seguem o mesmo critério de posicionamento.

Para o último exemplo, a semente é o módulo M04 (+), pois possui o maior número de conexoes (3 conexoes). O próximo elemento seria o 2,3 ou 5, todos com 2 conexoes. Pela ordem é selecionado o elemento 2, e o posicionamento fica da seguinte forma:

posic: p1[M02], p2[M04], p3[ ], p4[ ], p5[ ]

O próximo elemento a ser posicionado, pela ordem anterior, é o M03:

posic: p1[M02], p2[M04], p3[M03], p4[ ], p5[ ]

Então é posicionado o M05:

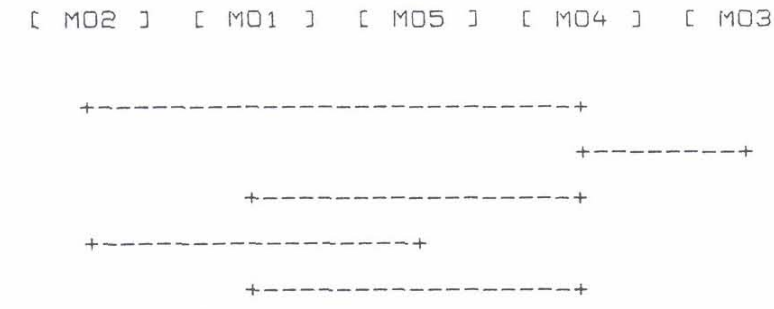
posic: p1[M02], p2[M05], p3[M04], p4[M03], p5[ ]

Note-se que este elemento pode trocar de posição com o elemento 4. Entretanto a solução apontada foi a primeira encontrada pelo algoritmo. Com isso, deseja-se enfatizar que, como melhora do algoritmo, pode-se agrupar as soluções que geraram um mesmo escore e prever nas próximas N situações qual seria a mais interessante. Por último é posicionado o elemento 1, pois possui apenas uma conexão, ficando a estrutura final assim:

posic:p1[M02],p2[M01],p3[M05],p4[M04],p5[M03]

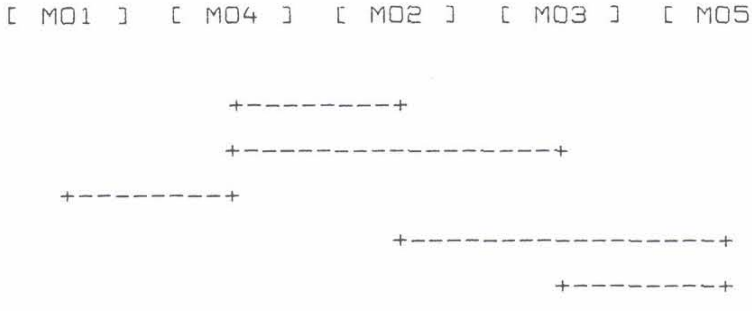
Deve ser ressaltado também que, embora o lugar dos elementos já posicionados não seja definitivo, a sua ordem deve ser respeitada.

Muito embora a solução não seja a ótima (o que é em métodos construtivos ou iterativos, já que o problema é NP-completo), chegou-se a uma solução razoável, como mostra o diagrama de relacionamentos seguinte:



Esta solução exige no mínimo 3 barramentos para ser implementada.

Comparada com uma segunda solução realizada manualmente:



Chega-se a conclusão que o número de barramentos pode

baixar para 2. Esta solução pode ser alcançada através do refinamento por iterações, como será analisado a seguir.

5.2.2 Posicionador Iterativo

Este algoritmo de posicionamento caracteriza-se pela simplicidade. A idéia é combater os prováveis problemas de um posicionamento inicial, como pouca vizinhança entre módulos, excesso de comprimento de conexoes, etc. Uma causa comum destes problemas é a existência de alguma conexão excessivamente longa. Pois bem, descoberta esta conexão, pode-se eliminá-la trocando de lugar algum elemento relacionado a esta conexão. O outro elemento a ser trocado é escolhido randomicamente. Caso o posicionamento seja melhor que o anterior segundo algum critério, este será o novo posicionamento. Então é realizada, da mesma forma, uma nova iteração, até chegar ao tempo limite.

Temos então três critérios, como na fig. 5.9:

- menor máxima distância : é escolhida a solução que possua um mínimo comprimento máximo de suas interconexoes, em relação as outras soluções obtidas.

- menor soma de interconexões : é escolhida a solução que apresente a menor soma total do comprimento de todas interconexões existentes.

- maior vizinhança : é escolhida a solução que possua maior número de vizinhos, ou seja, elementos conectados lado-a-lado diretamente.

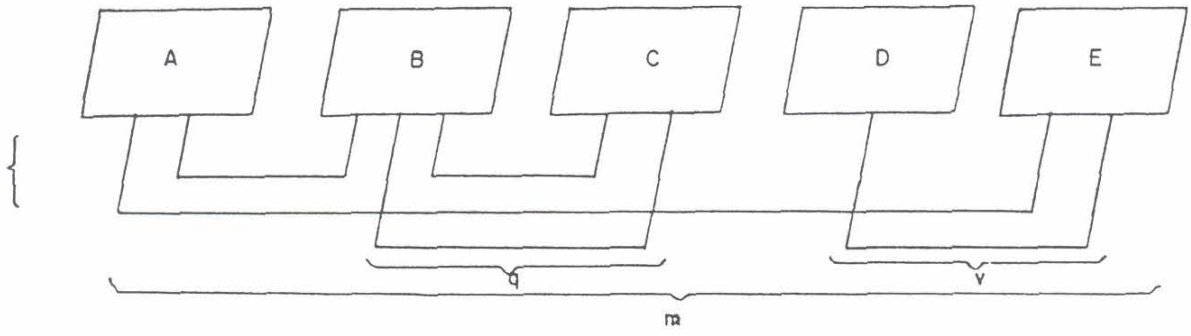


Figura 5.9 Critérios de Posicionamento

Cada um destes critérios levará a uma velocidade máxima de trabalho (determinado pelo menor comprimento máximo). O critério de menor máxima distância é o que influi mais neste caso. A idéia é permitir que o usuário escolha algum critério para o posicionamento ou pesos diferentes para os três critérios.

O exemplo anterior foi testado para um tempo de 5 segundos, obtendo-se os seguintes resultados:

- critério 1:

```
posic:    p1[ M01 ],p2[ M02 ],p3[ M03 ],p4[ M04 ],p5[ M05 ]

                        +--------------------+
                                  +----------+
              +----------+
                                             +----------+
                                  +--------------------+
```

Pode-se perceber que o comprimento da máxima conexão diminuiu.

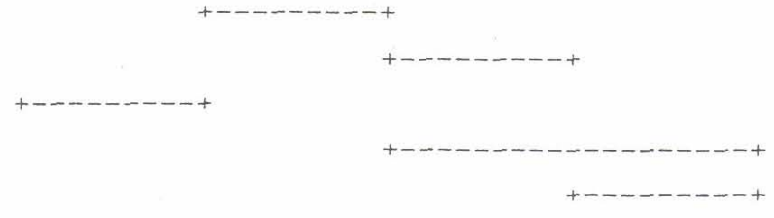
- critério 2:

posic: p1[MO1],p2[MO2],p3[MO3],p4[MO4],p5[MO5]

Mesma solução que solução manual.

- critério 3:

posic: p1[MO1],p2[MO2],p3[MO4],p4[MO3],p5[MO5]



A solução encontrada é interessante, pois possue uma soma no comprimento das rotas menor que a solução manual. O número de MOs vizinhos encontrada também é grande. Conforme a complexidade do problema, pode-se chegar a soluções semelhantes para diferentes critérios.

5.3 Roteamento e Cálculo do Número de Barramentos

Após o posicionamento dos diversos módulos operacionais, é necessário definir os conectores que estarão

envolvidos no roteamento, bem como sua disposição física, para que o roteador possa exercer sua função.

Para definir os conectores e suas ligações é utilizado um conjunto de fatos que, associados a algumas regras básicas, determinam uma solução adequada. A identificação de cada conector que fará parte da estrutura final são:

- número do conector,
- tipo (entrada ou saída),
- ordem do conector (primeira entrada, segunda saída, etc.).

Por exemplo, a operação R1 <- R2 + R3 exige um banco de dados com as seguintes informações para cada conector:

formato: conector (número, símbolo, tipo, ordem).

```
conector(1,R1,e,1).
conector(2,R2,s,1).
conector(3,R3,s,1).
conector(4,+,s,1).
conector(5,+,e,1).
conector(6,+,e,2).
```

Uma vez conhecidos os conectores necessários, deve-se conhecer as relações entre eles. A saída do somador deve ser conectada à entrada de R1, expresso da seguinte forma:

```
ligar(1,4,2).
```

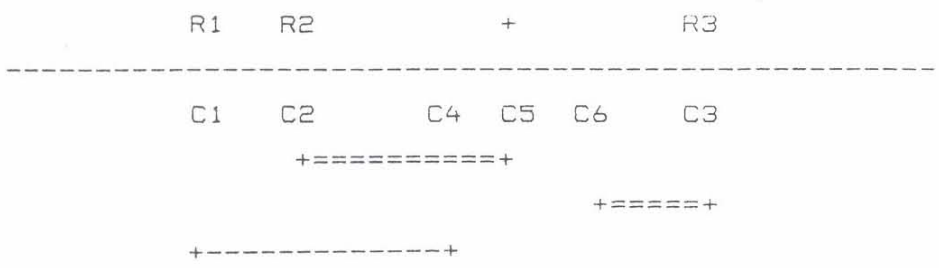
O fato quer dizer: conectar o conector 1 ao conector 4 na segunda fase.

formato: ligar (conector 1, conector 2, fase).

Desta forma obtem-se todas as conexoes necessárias para o exemplo:

```
ligar(2,5,1).
ligar(3,6,1).
ligar(1,4,2).
```

Pode-se expressar graficamente o problema da seguinte forma:



As rotas +---+ são executadas na fase 2 e as rotas +===+ são executadas na fase 1.

Cada rota indicada pelo fato ligar, deve ocupar um barramento. Rotas de diferentes fases podem ocupar a mesma posição em um barramento, já que acontecem em tempos diferentes. Para encontrar o número mínimo de barramentos, deve-se utilizar o algoritmo "left-edge" [WES85] que encontra uma solução ótima para o problema. Neste algoritmo as conexões são alocadas no barramento da esquerda para a direita. Quando não é mais possível utilizar o mesmo barramento, um novo barramento é utilizado, como no exemplo da fig. 5.10.

A fig. 5.11 mostra o barramento final e as chaves de barramento. As chaves de barramento separam conexoes que são executadas na mesma fase e que ocupam o mesmo barramento.

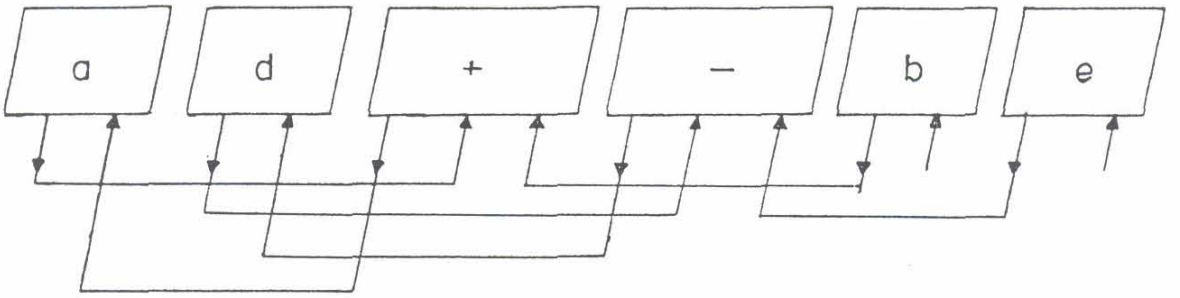


Figura 5.10 Algoritmo "left-edge" para roteamento

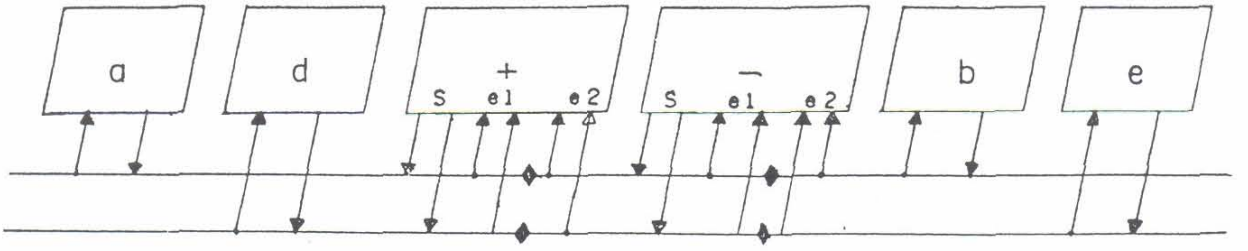


Figura 5.11 Escolha de Chaves de Barramento

Pode-se ver pela figura que as chaves de barramento ocupam posições assimétricas nos barramentos, e que existe uma chave interna no registrador E.

A rotina de roteamento do CSAR calcula:

- Número final de barramentos da arquitetura,

- Conectores que são ligados a cada barramento,

- Posição e número de chaves de barramento (este cálculo é importante para a parte de controle).

Na geração do layout são utilizados dois algoritmos conhecidos: um algoritmo de posicionamento e um algoritmo de roteamento. Este fato facilita a síntese automática e permite a utilização de técnicas já conhecidas e estudadas.

5.4 Arquitetura da Parte de Controle

Existe uma série de estruturas passíveis de serem utilizadas na implementação da máquina de estados do CSAR, já existentes no laboratório GME, como o Trago[MOR89] e o Tramo[LUB90]. Apresenta-se também outra solução para o problema, que é a de utilizar um PLA.

A utilização de Arrays Lógicos Programáveis (PLAs) como parte de controle adapta-se bem às técnicas de síntese automática. A escolha de PLAs para parte de controle do CSAR apresenta as seguintes vantagens:

- Simplicidade da estrutura.

- Diversificação quanto à topologia. A entrada e saída de sinais pode ser realizada em diversos pontos da estrutura.

- Modularidade. O PLA é constituído geralmente de quatro à seis tipos diferentes de células.

- A avaliação de desempenho e o dimensionamento de buffers apresenta farta bibliografia.

A maior desvantagem do PLA está na queda de desempenho quando aumenta a complexidade das máquinas de estados. O número de monômios diferentes implica em uma maior altura do PLA e portanto um maior atraso nos sinais de saída. Pode-se resolver o problema de aumento da complexidade da máquina de estados, dividindo-a em diversos níveis hierárquicos de controle. Cada nível hierárquico de controle é implementado através de um PLA. Esta solução, apesar de bastante interessante, foi descartada, pois sua complexidade exigiria um tempo maior de desenvolvimento.

A solução adotada para gerar a máquina de controle do CSAR foi utilizar um único PLA. Esta solução adapta-se a classe de problemas e circuitos que se deseja implementar com o CSAR. O PLA utilizado na arquitetura do CSAR sofreu algumas modificações topológicas, com relação aos PLAs tradicionais. O circuito implementado pelo PLA é do tipo Máquina de Moore, devido as características da linguagem de descrição do circuito e da lógica de seqüenciamento das operações.

5.4.1 PLA monomatriz

O PLA utilizado no CSAR é um PLA monomatriz. Este PLA apresenta como característica o fato de não estarem topologicamente separadas as matrizes E e OU, encontradas em outros PLAs, como mostra a fig. 5.12.

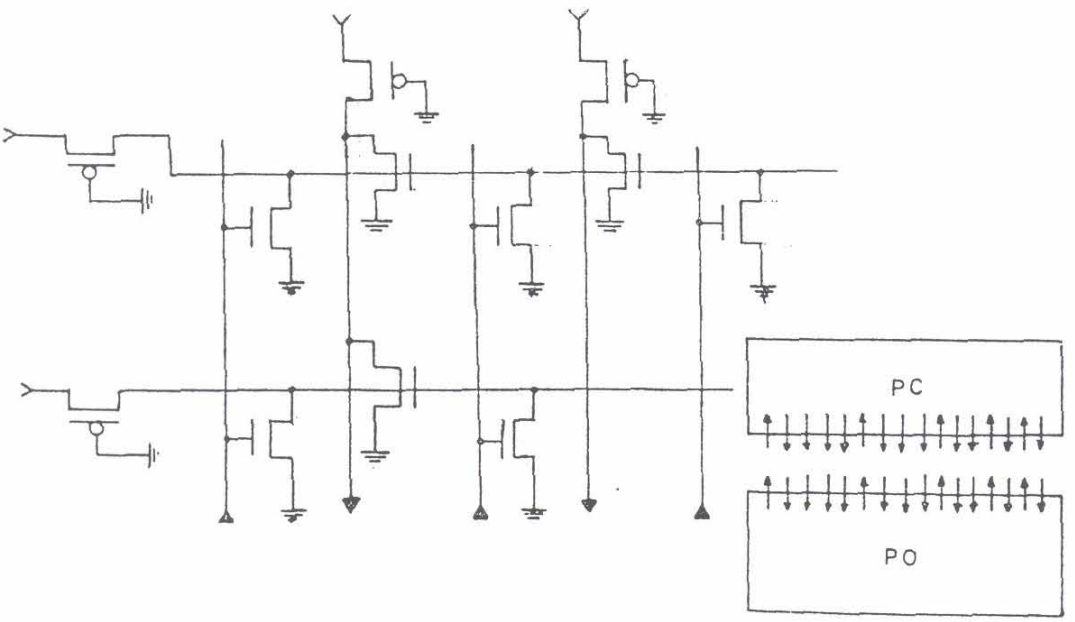
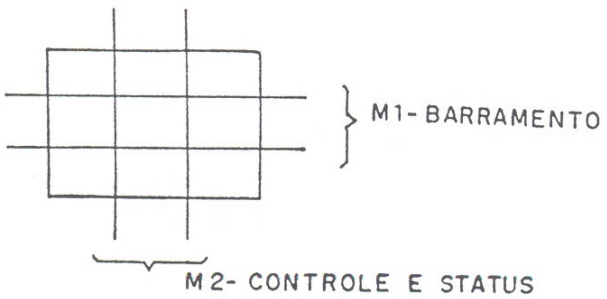


Figura 5.12 PLA Monomatriz

A vantagem de utilizar este PLA reside em que os sinais de entrada e saída poderem ser multiplexados. Como a arquitetura da parte operativa possui os sinais de entrada e saída nesta forma, devido as características topológicas da arquitetura, pode-se atingir um encaixe adequado das linhas dos sinais, como mostra a fig. 5.13.



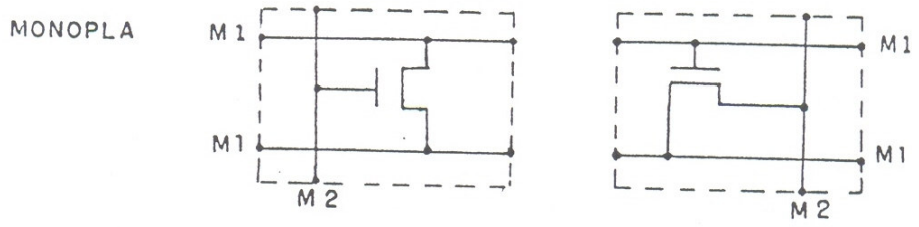


Figura 5.13 Ligação do PLA com a Parte de Controle

O diagrama elétrico das células do PLA monomatriz na fig. 5.14 mostram que os sinais que transportam as informações de entrada e saída utilizam apenas a camada de alumínio, o que aumenta a velocidade do PLA.

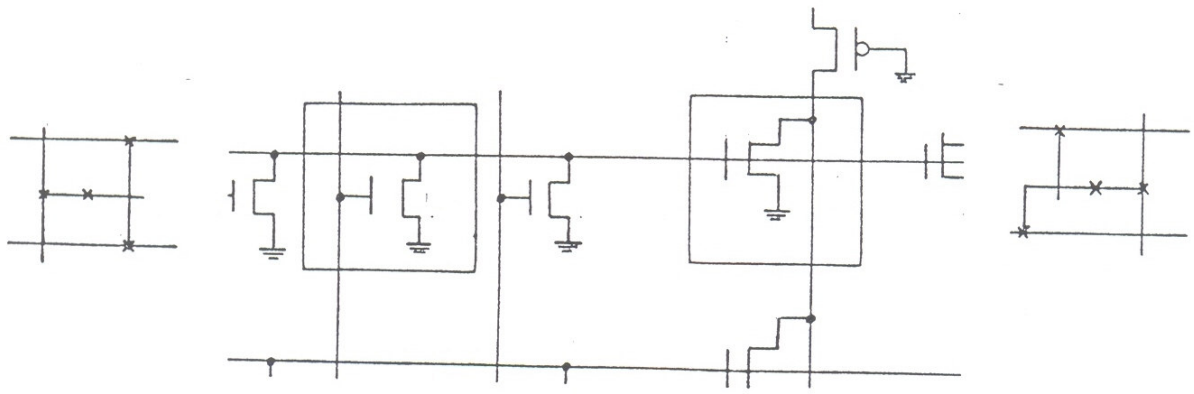


Figura 5.14 Diagrama Elétrico das Células do PLA

A utilização do PLA monomatriz é um interessante assunto para pesquisa. Como exemplo, pode-se sugerir aplicar o PLA monomatriz internamente a parte operativa, como na fig. 5.15. Com isso, a atraso de propagação dos sinais do PLA pode ser diminuído, além de chegar-se a uma solução bastante compacta.

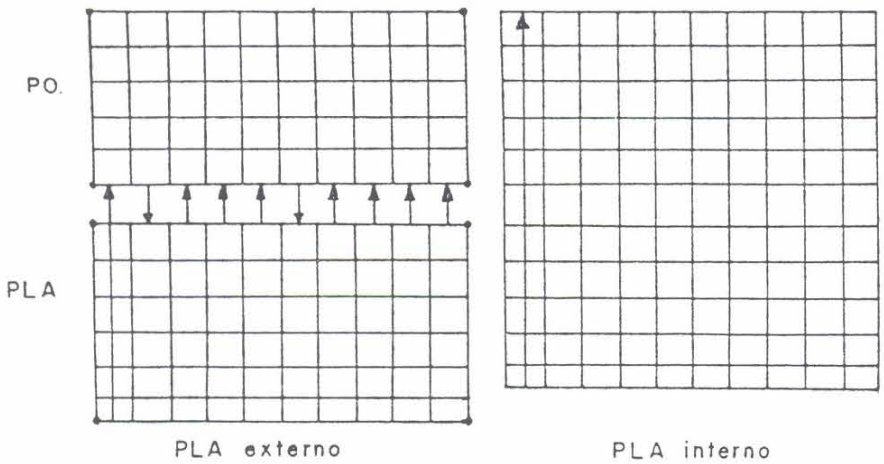


Figura 5.15 Mistura do PLA à Parte Operativa

5.4.2 Sinais de Controle

Os sinais de entrada e saída do PLA são responsáveis pelo correto funcionamento da parte operativa. Os que saem do PLA podem ser:

- Habilitação de Escrita no Barramento (HE).

Este sinal permite que as saídas dos registradores e saídas dos módulos operacionais sejam escritas nos barramentos a que estão ligadas.

- Habilitação de Leitura do Barramento (HL).

Este sinal habilita que as entradas dos registradores e dos módulos operacionais recebam o conteúdo do barramento a que estão ligadas.

- Habilitação das Chaves de Barramento (HC).

Este sinal liga e desliga as chaves de barramento. A velocidade dos sinais de habilitação de chaves de barramento deve ser maior que a velocidade dos sinais de habilitação de escrita e leitura de barramento. Para isso é necessário que os buffers do PLA sejam dimensionados de forma correta.

- Estado Calculado da Máquina de Estados (PE).

Este sinais são utilizados no cálculo do próximo estado do PLA. Tratando-se de uma máquina de estados do tipo Moore (as saídas são funções apenas do estado atual), os sinais de estado atual são diretamente realimentados para o PLA e não são utilizados pela parte operativa.

Os sinais de entrada do PLA são os seguintes:

- Estado Atual da Máquina de Estados (EA).

Utilizado no cálculo do próximo estado.

- Saída dos Módulos Operacionais (MO).

As saídas dos módulos operacionais que entram no PLA são os resultados de comparações, que influenciam no cálculo do próximo estado.

O PLA implementa portanto as seguintes equações:

PE = f (EA, MO)
HE, HL, HC = f (PE)

O CSAR deve encontrar todos os sinais HE, HL e HC e calcular sua equação em função do PE.

5.4.3 Minimização Booleana

A solução utilizando PLA pode ser otimizada se existir uma algoritmo de minimização booleana das equações a serem implementadas no PLA. Como a altura do PLA é proporcional ao número de monômios das equações, a redução de um único monômio já acarreta uma economia de área significativa.

O CSAR possui um algoritmo de minimização booleana que funciona da seguinte forma:

- Inicialmente a equação é colocada na forma canônica. Este formato é simples de ser obtido, pois as saídas dependem apenas do estado atual.

- Todos os termos da equação são então combinados dois a dois. Se os termos diferirem apenas em uma variável, então estes termos são simplificados.

- Se algum termo não foi envolvido em alguma simplificação então este termo não pode mais ser simplificado.

- Neste ponto é feita uma avaliação de todas as equações, em conjunto, e determinado o número total de termos diferentes entre si. Se o número de termos for o menor número de termos até o momento, então a equação é armazenada.

- A nova equação simplificada deve continuar as operações anteriores de otimização até que não seja mais possível nenhuma otimização.

No final do algoritmo, as equações escolhidas para implementar a máquina de estado são aquelas cuja soma de termos diferentes foi a menor.

O algoritmo implementado utiliza os passos iniciais da técnica proposta por McCluskey [WES85].

O princípio da combinação de termos pode ser melhor compreendido no seguinte exemplo:

S = AB + aB + Ab + ab

Onde: A = not a / B = not b.
A equação pode ser reescrita da seguinte forma:

S = (AB + aB) + (AB + Ab) + (AB + ab) +
(aB + Ab) + (aB + ab) + (Ab + ab)

Esta forma é a combinação de todos os termos da primeira forma. A repetição de termos é válida, pois:

A + A = A ou a + a = a

Com a primeira minimização (resolvendo as somas entre parênteses) chega-se na seguinte equação:

S = B + A + a + b

Refazendo as combinações:

S = (B + A) + (B + a) + (B + b) +
(A + a) + (A + b) + (a + b)

E, finalmente:

S = (B + A) + (B + a) + 1 + 1 + (A + b) + (a + b) = 1

O algoritmo chega sempre a solução ótima para a menor equação, pois cobre todas as combinações possíveis. Entretanto o algoritmo não atinge garantidamente a melhor solução para um PLA, embora possa diminuir o seu tamanho.

As seguintes equações,

S1 = AB + ab + aB
S2 = AB + ab

apresentam um total de três termos diferentes, que formarão um PLA de três termos intermediários (proporcional à altura).

Se as equaçöes forem minimizadas chega-se a

S1 = B + a
S2 = AB + ab

Esta solução, apesar de apresentar equaçoes mais otimizadas, apresenta quatro termos diferentes, e implica em um PLA maior (30 %) do que a primeira solucão. Além da escolha do algoritmo a ser implementado para minimização booleana, é importante para obter-se uma solução em tempo razoável que a estrutura de dados seja corretamente escolhida.

Um algoritmo aparentemente interessante pode fracassar se a escolha da estrutura de dados não corresponder às espectativas. Por exemplo, descobrir se dois termos são vizinhos, ou seja, diferem apenas em uma variável deve ser feito da forma mais rápida possível. A melhor alternativa é comparar todas as variáveis de forma simultânea, o que pode ser feito se os termos forem representados por dois bytes:

1# byte - valor das variáveis.
2# byte - existência das variáveis na equação.

Assim, a equação:

S = ABcD + aBC

pode ser representada da seguinte forma:

1# termo: 1# byte 00001011
2# byte 00001111

O primeiro byte indica que a variável A (bit 0), variável B (bit 1), variável c (bit 3) e variável D (bit 4) fazem parte da equação.

O segundo byte indica que as variáveis A,B,C e D possuem algum significado na equação.

Para o segundo termo temos:

1# termo: 1# byte 00000110
2# byte 00000111

A comparação de dois termos é feita pela operação OU-EXCLUSIVO do 1# byte e pelo AND do 2# byte. Se o 2# byte dos termos for diferente, os termos não podem ser vizinhos. Esta estrutura de dados é utilizada no CSAR. O programa que implementa o algoritmo proposto encontra-se nos anexos.

5.5 Análise de Irregularidades da Arquitetura

O processo de síntese automática deve prever a existência de irregularidades ou estruturas heterogêneas na arquitetura. Exemplo de irregularidades são: entrada e saídas de sinais conectados aos pads, cálculo de "carry look ahead", operações com fatias ou conjunto de bits dos registradores, tamanho de registradores, etc.

A solução de algumas irregularidades não é muito simples, envolvendo em alguns casos a utilização de técnicas específicas encontradas em muitas ferramentas de CAD para microeletrônica. A conexão de sinais aos pads, por exemplo, pode ser feita por uma ferramenta de roteamento automático de canal.

5.5.1 Entrada e Saída de Sinais

Os sinais de entrada e saída de dados são sinais de registradores que são ligados diretamente aos pads (logo, não são conectados aos barramentos).

Como os sinais dos registradores serão conectados aos pads, é interessante posicionar os registradores nas extremidades do "bit-slice". Caso não seja possível posicionar algum registrador de entrada e saída nas extremidades do "bit-slice" pode-se adotar duas soluçoes: utilizar uma linha de transparência da célula (criando um novo barramento) ou abrindo o "bit-slice" e roteando os sinais para a interface externa da parte operativa (fig. 5.16).

5.5.2 Cálculo de Carry Look Ahead

Apesar de o somador ser dividido em um conjunto de células idênticas, e portanto possuir bastante modularidade, a inserção de uma lógica de "carry look ahead" buscando uma maior velocidade de propagação do carry, pode gerar um conjunto de irregularidades na arquitetura.

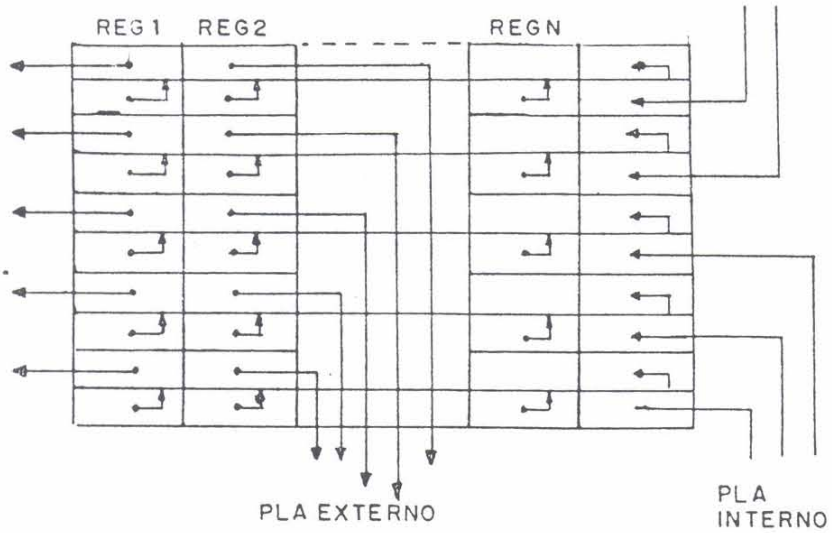


Figura 5.16 Roteamento dos Registradores aos PADs

Se a lógica de aceleração do carry é inserida junto a alguns dos bits do somador, os bits que não possuem a lógica terão uma área desperdiçada (fig. 5.17).

Se a lógica de aceleração do carry é inserida entre dois bits do somador, portanto ocupando um bit slice, (fig. 5.18), o restante da área do bit-slice que não for ocupada será desperdiçada.

A escolha da melhor solução nem sempre é uma tarefa simples para um Compilador de Silício, pois deve atender tanto as especificações do usuário quanto ao desempenho desejado. Dentro da filosofia do CSAR, cabe ao usuário decidir qual a solução mais conveniente. Para isso é

necessário avaliar os resultados que podem ser obtidos pelas duas soluções.

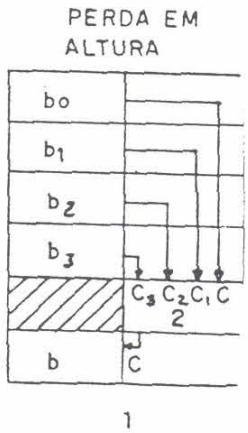


Figura 5.17 Primeira solução para aceleração do carry

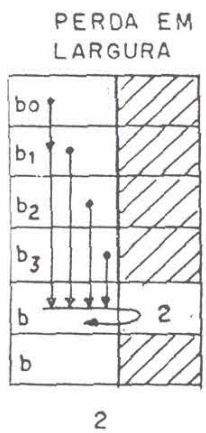


Figura 5.18 Segunda solução para aceleração do carry

5.5.3 Operações com Fatias de Palavras

A linguagem de descrição OPCODE prevê a existência de operações com fatias de registradores, ou seja, operaçoes que englobam alguns bits dos registradores.

A seguinte operação, por exemplo:

A[2:4] <- B[0:2] ;

A operação descrita pode ser resumida em:

bit 2 do registrador A = bit 0 do registrador B
bit 3 do registrador A = bit 1 do registrador B
bit 4 do registrador A = bit 2 do registrador B

Quando não existe nenhuma descrição de quais bits estão envolvidos na operação, é assumido por "default" que todos os bits do registrador estão envolvidos. Para que este tipo de operação seja realizado em uma estrutura bit-slice, é necessário que bits de um bit-slice possam ser utilizados em outro bit-slice. Neste caso, pode-se realizar a transferência de bits por um canal aberto no bit-slice, como mostra a fig. 5.19.

Uma segunda solução é utilizar registradores deslocadores que realizam o deslocamento antes da transferência de registradores e após a execução da operação (fig. 5.20).

A primeira solução foi a escolhida, pois apresenta um desempenho muito superior quanto a velocidade de execução,

além da área ocupada ser equivalente (pois dispensa a utilizaçåo de registradores deslocadores).

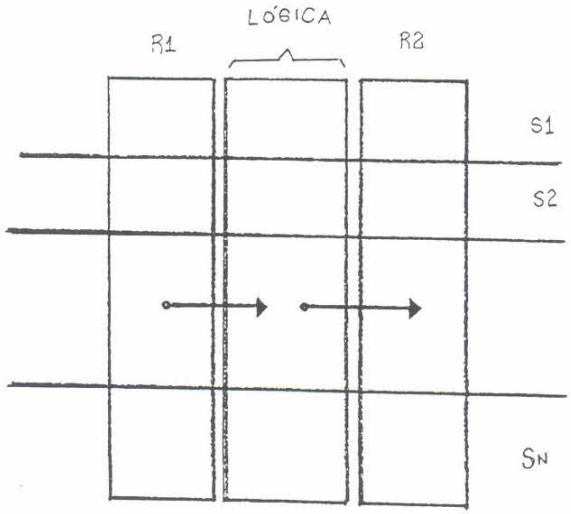


Figura 5.19 Primeira soluçåo para fatia de palavras

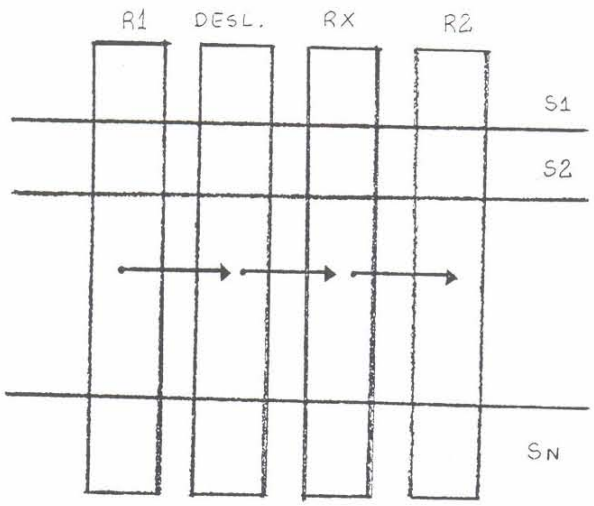


Figura 5.20 Segunda soluçåo para fatias de palavras

5.5.4 Tamanho dos Registradores

O tamanho dos registradores nem sempre é idêntico. Se alguns registradores são maiores que outros duas soluçoes podem ser adotadas: manter cada bit do registrador em um bit-slice ou dividir o registrador, ocupando dois bits ou mais por bit-slice (fig. 5.21).

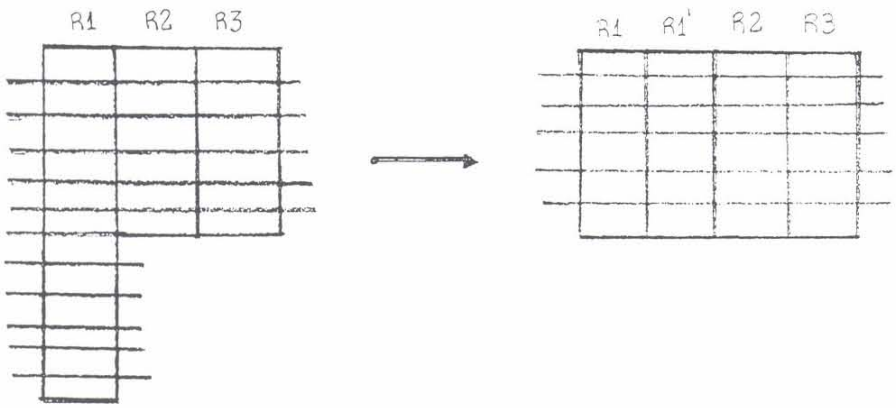


Figura 5.21 Diferentes tamanhos de registradores

No CSAR o usuário deve especificar o número máximo de bit-slices. Se o usuário especifica um número máximo de bit-slices de 4 e utiliza registradores de 16 bits, então os bits estarão divididos nos bit-slices da seguinte forma:

bit-slice 1: bits 0,4,8 e 12.
bit-slice 2: bits 1,5,9 e 13.
bit-slice 3: bits 2,6,10 e 14.
bit-slice 4: bits 3,7,11 e 15.

Esta escolha de distribuição dos bits facilita as operações com fatias de palavras, já que estas só podem ser especificadas com seqüências de bits vizinhos.

Se o usuário especificou, por exemplo:

Registrador A - 8 bits.
Registrador B - 4 bits.
Número de Bit-Slices - 4.

#1: A[2:5] <- B[0:3] / #1 ;

A operação no estado #1 poderá ser executada normalmente, pois os barramentos ficaram assim divididos:

bit-slice 1: A[0]-A[4] e B[0].
bit-slice 2: A[1]-A[5] e B[1].
bit-slice 3: A[2]-A[6] e B[2].
bit-slice 4: A[3]-A[7] e B[3].

Ou seja, os valores de A[2] à A[5] se encontram em diferentes bit-slices.

Outra característica interessante da arquitetura do CSAR é que os bits pertencentes ao mesmo registrador e que se encontram em um mesmo bit-slice não necessitam

obrigatoriamente serem vizinhos.

Para o exemplo anterior, a fatia A[2:5] e a fatia A[6:1] (note-se que A[6:1] = { A[6] , A[7] , A[0] , A[1] }) são consideradas como dois registradores diferentes.

5.6 Modificações na Linguagem OPCODE

A escolha da arquitetura alvo permite que algumas particularidades sejam acrescentadas a linguagem de entrada OPCODE. Contudo, a base da linguagem é mantida, e independe da arquitetura alvo escolhida. A primeira especificação é quanto a diferença de tempo de execução de algumas operações.

A atribuição, por exemplo, é executada em um único ciclo de relógio. Já as operações de comparação, aritmética e lógica necessitam de dois ciclos de relógio. Um ciclo de relógio é composto por duas fases: fase1 e fase2. Portanto, operações de comparação, aritmética e lógica utilizam quatro fases de relógio:

Ciclo 1:
fase 1 / fase 2

Ciclo 2:
fase 1 / fase 2

Uma operação de atribuição de registradores pode ser executada no ciclo 1 ou no ciclo 2.

Este ciclo deve ser especificado da seguinte forma:

Registrador Destino <- Registrador Fonte (ciclo 1/2)

Ex:
#1: R1 <- R2 (ciclo 1) / R2 <- R3 (ciclo 2)

Neste caso as duas operações podem compartilhar do mesmo barramento, já que são executadas em ciclos diferentes. Cabe ao usuário especificar qual o ciclo será utilizado para as operações de atribuição.

Se uma operação de atribuição não tiver especificado qual ciclo será realizada a operação, é assumido por "default" o ciclo 1.

5.7 Verificação e Validação

Para assegurar que a descrição do circuito a ser gerado pelo CSAR é correta, bem como a arquitetura gerada, deve-se estudar métodos de verificação e validação da arquitetura. Por verificação entende-se assegurar que todos conectores e todos os sinais estão corretamente ligados, tanto na parte operativa como na parte de controle. Já a validação da descrição e da arquitetura deve garantir que a solução adotada e a forma como foi adotada podem realmente sintetizar circuitos que funcionam como esperado.

A verificação dos resultados da ferramenta baseia-se na verificação das diversas etapas ou níveis de transformações por que passa a descrição do circuito até atingir o nível de

máscaras. Os algoritmos desenvolvidos na implementação do CSAR poderão ser verificados com o uso contínuo da ferramenta e análise dos resultados obtidos.

Por enquanto os maiores testes foram realizados com os algoritmos individualmente, que são aplicados em diversos níveis de descrição da ferramenta. Já a validação da ferramenta pode ser feita através de simulaçoes lógicas e elétricas do comportamento do circuito. Entretanto a validação da arquitetura sempre continuará dependente da fabricação e testes de um circuito totalmente gerado pelo CSAR.

Mesmo assegurando que um circuito esteja correto (ou seja, passou corretamente pela verificação e validação) é necessário realizar testes no circuito com o objetivo de encontrar possíveis falhas. As falhas que podem ocorrer são tipicamente decorrentes do processo de fabricação. Como no CSAR a validação dependerá do desempenho dos testes com amostras concebidas, é necessário separar bem os problemas de validação do circuito das falhas do processo de fabricação.

No teste de um circuito, a visualização de apenas alguns sinais externos é insuficiente na localização de falhas ou erros. Neste caso é interessante que se possa testar individualmente o PLA, os registradores e os módulos operacionais.

Para tanto, é implementado junto a compilação da parte de controle e da parte operativa um estrutura de testes composta por um registrador deslocador, como é mostrado na fig. 5.22. O registrador deslocador possui um registrador

para cada sinal de entrada ou saida do PLA.

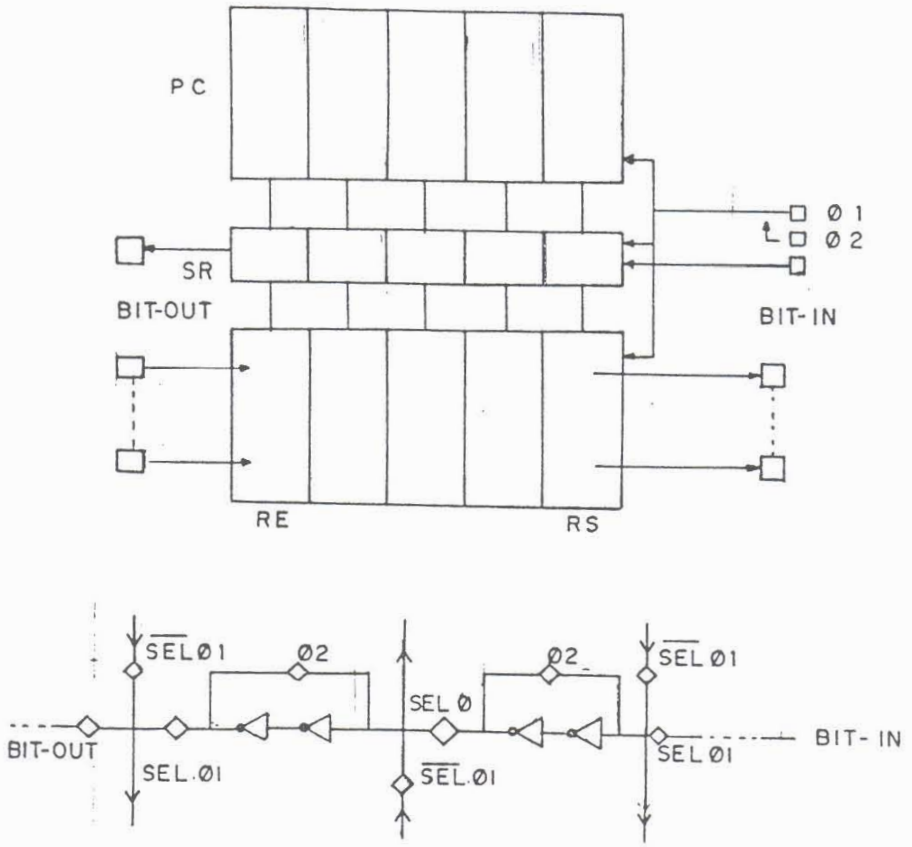


Figura 5.22 Estrutura de teste

O registrador deslocador é controlado por quatro "pads" com as seguintes funçoes:

-pad 1 : sinal de deslocar.

-pad 2 : entrada do registrador de deslocamento.

-pad 3 : saida do registrador de deslocamento.

-pad 4 : seleciona se o sinal que entra no registrador é

externo ou interno. Se for interno, corresponde ao bit anterior na seqüência de deslocamento.

Os procedimentos de teste seguem a seguinte seqüência:

- Inicialmente é feito o teste do registrador deslocador, inserindo-se bits na entrada do registrador e após sucessivos deslocamentos verificando a saída dos bits escritos.

- Os registradores de entrada e saída que são ligados ao pads de entrada e saída devem ser testados. Para isso, pode-se escrever um dado em um registrador de entrada, transmitir o dado através do barramento para um registrador de saída e finalmente ler o conteúdo do registrador de saída.

- Testar os demais registradores, seguindo um processo semelhante aos testes anteriores.

- Testar os Módulos Operacionais.

- Testar as Equações do PLA.

Após a realização dos testes pode-se garantir que as estruturas funcionam individualmente, principalmente a parte operativa e a parte de controle.

6. REPRESENTAÇÃO SIMBÓLICA

6.1 Introdução

Continuando a apresentação e definição da ferramenta de software CSAR, este capítulo mostra as soluções encontradas pelo autor para resolver o problema de dependência tecnológica.

O CSAR utiliza um programa de DRC (já desenvolvido) como expansor simbólico. Esta nova ferramenta de software (CORCEL) que foi desenvolvida pelo autor para diversas aplicações é utilizada pelo CSAR, para expansão simbólica visando independência de tecnologia.

6.1 Independência de Tecnologia

O crescente surgimento de novas tecnologias de fabricação de circuitos integrados, bem como o aprimoramento das tecnologias existentes, geram um sério problema de atualização das regras de projeto de circuitos, blocos funcionais e células de uma biblioteca, assim como das ferramentas de síntese de layout.

Quando os projetos são realizados satisfazendo as regras de projeto de uma única tecnologia, podem surgir problemas como: aprimoramento da tecnologia com valores de regras menores, troca de fabricante ("Silicon Foundry") ou de processo . A solução habitual para qualquer um destes problemas é reprojetar as células existentes, adaptando às novas regras.

Esta solução implica necessariamente em uma grande perda de tempo, pois a atualização de um conjunto de células de uma determinada tecnologia é geralmente um trabalho manual. Em alguns casos, é mais fácil redesenhar novamente todas as células.

A solução imediata para este problema é automatizar e parametrizar o processo de edição das máscaras. Entretanto, esta solução defronta-se com os costumeiros problemas da síntese automática: área e velocidade final do circuito. A geração automática a partir de um nível esquemático, por exemplo, mostra-se ainda pouco eficiente quando comparada às soluções obtidas manualmente.

A experiência obtida na área de projetos mostra que antes de realizar o projeto das máscaras de uma célula o projetista realiza geralmente um diagrama de "stick" [MEA80],[WES85]. O diagrama de "stick" tornou-se realmente muito útil para obter-se soluçoes para o problema de independência de tecnologia. Este diagrama consiste de linhas retas que representam as camadas utilizadas na ediçăo das máscaras (fig. 6.1).

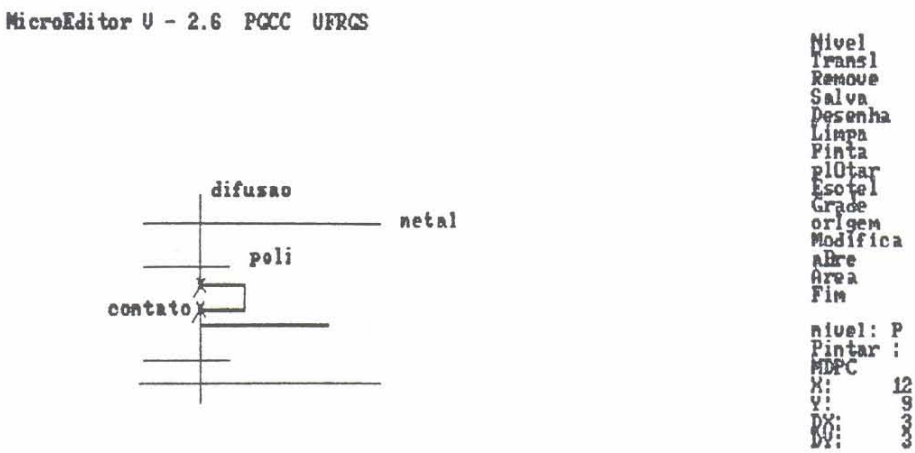


Figura 6.1 Diagrama de Stick

A diferença básica de uma ediçăo em stick, com relaçăo a ediçăo de máscaras em retângulos está no fato de que uma grande parte das regras de projeto năo necessitam ser respeitadas. Um exemplo típico destas regras săo: distâncias entre camadas, largura mínima de camadas, fatores de mérito dos transistores, dimensăo de contatos e vias, etc. Para o projetista, a utilidade deste diagrama está na possibilidade de realizar um planejamento da posiçăo dos elementos ativos

do circuito (transistores) e de que forma será possível interligá-los, utilizando os níveis de máscaras disponíveis.

A fig. 6.1 mostra o diagrama de "stick" de um "transmission gate". Este mesmo diagrama pode ser realizado através de um editor de máscaras, como mostra a fig. 6.2. Pode-se chamar este último diagrama de um diagrama de barras.

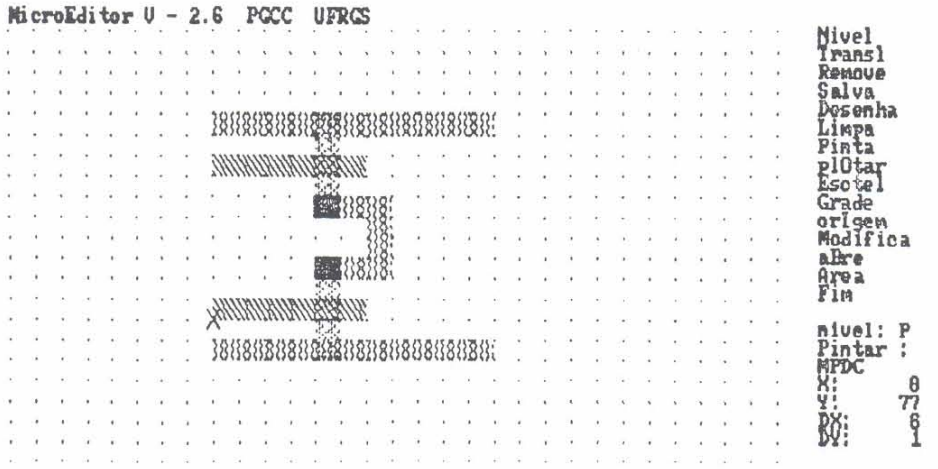


Figura 6.2 Diagrama de Stick em um Editor de Máscaras

Note-se que não há preocupações com as diversas classes de regras de projetos já enumeradas. Existe apenas uma preocupação com o posicionamento dos transistores e de que forma eles podem ser interligados com as camadas existentes (como podem ser roteados utilizando contatos, difusões, metais ou polisilício). O nível de descrição em que se encontra o diagrama de barras é o nível normalmente referenciado como nível simbólico [WES85].

É possível considerar este nível de descrição como imediatamente superior ao nível de máscaras. Este fato, porém, não consiste em nenhuma regra pré-determinada. Outra alternativa seria considerar-se o nível imediatamente superior como um diagrama de barras onde as camadas já possuem sua largura final ou distância respeitando as regras de projeto. Todavia, a primeira escolha mostra uma característica bastante peculiar, que é a eliminação de várias regras de projeto e de vários problemas topológicos.

6.3 Editor de Barras

A idéia de utilizar um Editor de Barras através de um Editor de Máscaras tem o objetivo de aproveitar as ferramentas de CAD disponíveis no GME. O trabalho de desenvolver uma ferramenta com todas as características existentes no ME [JAC86] ou no EMHIR [JAC88] é desnecessário. Com isso tornou-se dispensável desenvolver uma ferramenta específica para edição de diagramas de barras. Os problemas que surgem desta opção podem ser resumidos em:

- O usuário deve especificar uma largura para todas as camadas. Esta largura pode ser mínima (um lambda), ou não. Geralmente nos Editores Simbólicos, não existe preocupação com largura mínima de camadas.

- A estrutura de dados gerada pelos Editores de Máscara é uma linguagem de descrição de retângulos. Estas linguagens visam a descrição final do layout do circuito (pois este é o seu objetivo) e carecem de uma série de facilidades para a

passagem do nível de descrição simbólico para o nível de máscaras. A especificação do fator de mérito de transistores, por exemplo, deve ser realizada através de modificaçoes nos editores de máscaras, ou através de inserçoes de comentários na descrição final.

Já as vantagens são resumidas em:

- Não é necessário desenvolver uma ferramenta especifica para a edição do diagrama "stick". É necessário apenas implementar algumas novas funções aos Editores de Máscaras já existentes. Acredita-se que esta vantagem será maior quando os Editores de Máscaras preverem caracteristicas típicas do nivel simbólico, como largura mínima das camadas e caracteristicas associadas aos transistores.

- Compatibilidade com outros Editores de Máscaras existentes, e aproveitamento de suas caracteristicas peculiares.

Para que a utilização do diagrama de barras seja possivel é necessário estudar a descrição gerada pelos Editores de Máscaras, que no caso do GME é a linguagem RS [TOD85], e extrair da descrição em retângulos informaçoes suficientes para atingir o nivel de máscaras.

5.3 Expansor Simbólico

A ferramenta que faz a passagem de uma descrição

simbólica para a descrição das máscaras será aqui chamada de Expansor Simbólico, embora esta passagem muitas vezes não envolva apenas o processo de expansão (como é o caso de algumas ferramentas que utilizam compactadores [WER82]). Existem diferentes tipos de expansões a serem realizadas sobre a célula original (desenhada através de um editor de barras). O tipo de expansão mais utilizado está baseado em uma descrição própria do nível de descrição simbólico [DUN80]. A partir desta descrição, as camadas são redesenhadas e especificadas com novas características, que evidentemente respeitam as regras de projeto [STA84]. O maior problema desta solução é a implicância que determinadas regras exercem sobre diversos pontos do circuito.

Pode-se ver na fig. 6.3 um exemplo tipico de perda de área na expansão. Considera-se uma mesma linha horizontal onde diversas camadas estão dispostas. Pode-se ver que após a expansão as regiões A e B estão praticamente inutilizadas, pois foram geradas pela necessidade de alargamento da camada que envolve o contato.

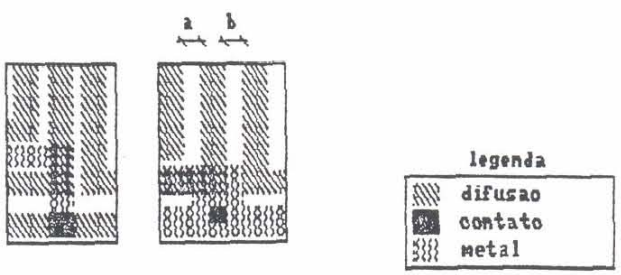


Figura 6.3 Perda de Area na Expansão

Para resolver este problema normalmente são utilizados compactadores [BOY88][CHO85] que são bastante eficientes (apesar do problema ser NP-completo).

Este expansor, que é baseado em um corretor automático de regras de projeto, funciona da seguinte forma. Após a edição do diagrama de barras, a célula passa pelo DRC que verifica os erros de projeto (na verdade todas as camadas estão envolvidas, neste instante, em algum erro de projeto, já que a célula está descrita em um nível simbólico). A medida em que os erros são encontrados eles são automaticamente corrigidos através de múltiplas expansões. As expansões aqui realizadas diferem bastante das realizadas pelos expansores analisados anteriormente. Isto porque elas estão concentradas apenas nas regiões do circuito onde foram encontrados erros pelo DRC.

Para melhor entendimento do processo, pode-se analisar a fig. 6.4 onde encontra-se um erro de distância entre duas camadas e deve-se abrir o circuito para sua solução. A abertura deste circuito é realizada apenas no trecho onde o erro de projeto existe.

O DRC utilizado para criar o Expansor Simbólico é o DARC [GOM88] desenvolvido no GME. A partir do DARC foi criado um novo programa, o CORCEL (Corretor de Células). O CORCEL é um programa que realiza a correção automática dos erros de projeto das células. O processo de correção segue os seguintes passos:

- Leitura do arquivo de descrição de uma célula (não hierárquico).

- Verificação das Regras de Projeto.

- Correção dos Erros de Projeto através de Expansoes.

- Gravação do Arquivo com as regras corrigidas.

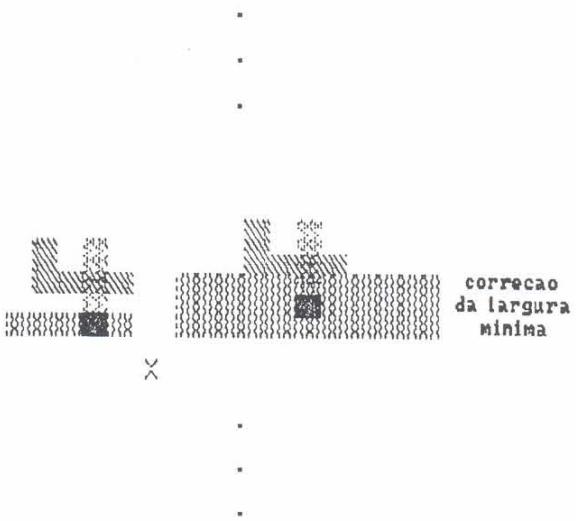


Figura 6.4 Exemplo de Abertura em uma Célula

A utilização do CORCEL como Expansor Simbólico é feita da seguinte forma:

- A célula de entrada pode ser uma célula editada no Editor de Barras.

- O fator de mérito desejado para os transistores é especificado como comentário no cabeçalho da descrição da célula.

- O CORCEL realiza a correção da célula. Como a célula está em um nível simbólico, existem diversas correçoes a serem realizadas.

- Após a correção da célula é atingido o nível de máscaras, com todas as regras de projeto respeitadas.

O algoritmo de expansão do CORCEL utiliza diretamente a estrutura de dados do DARC. Quando são encontrados erros de distância entre camadas ou de largura mínima de camadas, o circuito é aberto e as estruturas são deslocadas até que o erro de projeto desapareça. Durante a abertura do circuito, alguns cuidados devem ser tomados. Nem sempre é possível realizar a abertura do circuito em linha reta, sob pena de alterar o tamanho de alguns elementos como contatos, vias e transistores (fig. 6.5).

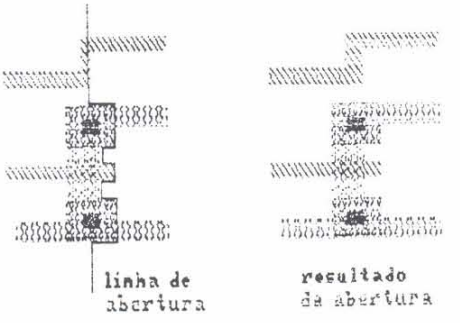


Figura 6.5 Abertura contornando Contatos e Transistores

Se o processo de expansão da célula for bem realizado, pode-se evitar a utilização de um compactador. O processo de edição simbólica geralmente requisita o uso de um programa compactador, para que as soluçoes obtidas sejam melhoradas. As soluções obtidas automaticamente, em muitos casos, ainda deixam a desejar quanto à área final.

7. PROGRAMA CSAR

7.1 Introducão

Este capitulo apresenta a interface com o usário do programa CSAR. São descritos os formatos dos arquivos de entrada e saida, e a escolha de linguagem de programação.

7.2 Especificações do Usuário

Os Compiladores de Silício devem ser abertos o suficiente para que o usuário possa especificar as características do circuito a ser gerado. Se for um circuito que irá operar a uma freqüência baixa, o usuário pode dar maior atenção para a área final que o circuito ocupará.

Da mesma forma, se o usuário precisa de um circuito que opere na máxima freqüência possível, a área final do circuito pode tornar-se uma especificação secundária ou de menor prioridade. O usuário ou projetista busca uma solução que está entre a solução de maior velocidade ou de menor área. Este fato é que torna a especificação do que realmente o usuário deseja difícil de ser traduzida em um conjunto de procedimentos a serem seguidos.

A solução encontrada no CSAR para permitir ao usuário uma maior interferência nos resultados da compilação foi permitir uma parametrização de diversas operações do Compilador de Silício.

Como exemplo de parametrização pode-se citar:

- Parametrização dos transistores,

- Escolha de critérios de posicionamento,

- Parametrização dos buffers de entrada e saída dos PLAs,

- Criação de células próprias pelo usuário,

- Modificações na descrição de entrada.

Entretanto o CSAR obriga ao usuário refazer a compilação se a solução encontrada não for a desejada.

7.3 Arquivos de Entrada e Saída

Os arquivos de entrada do CSAR são os seguintes:

- Arquivo de descrição do circuito na linguagem OPCODE.

- Arquivo com as células dos Módulos Operacionais (inclusive dos registradores) editadas utilizando o Editor de Barras (Biblioteca de Células).

No cabeçalho das células deve ser inserido um comentário com as seguintes informações:

- Nome da célula,
- Tipo da célula,
- Sinais ligados à parte operativa,
- Sinais ligados à parte de controle,
- Tamanho da célula,
- Símbolo da célula na descrição OPCODE.

Estas informações são descritas da seguinte forma:

```
{
CSAR Célula do Somador
TIPO Prim
SINL Entrada 20
PLAS s  wr 16
TAMH 120
SIMB [+]
}
```

O símbolo CSAR dentro de um comentário indica que os dados a seguir fazem parte da descrição da célula para entrada de dados do CSAR. O primeiro dado após o símbolo CSAR deve ser o nome ou descrição da célula (no caso, "Célula do Somador"). O símbolo TIPO indica qual bit da estrutura bit-slice está sendo representado:

prim - primeiro bit ou inicial.

par - bits pares após o primeiro bit e antes do último bit.

impar - bits ímpares após o primeiro bit e antes do último bit.

último - último bit da estrutura.

Após a leitura do cabeçalho de todas as células o CSAR faz um teste de consistência. Se alguma célula faltar na montagem da estrutura, o usuário é avisado da situação. O símbolo SINL informa se a célula é apenas a descrição da entrada de alguma célula, se é o corpo da célula ou se é a saída da célula.

entrada N - a célula descreve o circuito de entrada do somador, que deve ser conectado ao barramento na posição N.

saída N - a célula descreve o circuito de entrada do somador, que deve ser conectado ao barramento na posição N.

corpo - a célula descreve o corpo ou o circuito que realmente implementa a soma.

O símbolo PLAS indica que o sinal a seguir deve ser conectado ao PLA, é um sinal de saída do PLA (s ou e) de nome wr, e está na posição X = 16. O símbolo TAMH informa a largura da célula, que no caso é 120 lambdas. Outros sinais de PLA podem ser descritos utilizando o mesmo formato.

A última informação é o SIMB ou símbolo que diz qual o símbolo será utilizado na descrição do circuito. Se o símbolo for uma letra significa que a célula é de um registrador. Este tipo de descrição permite que o usuário especifique suas próprias células e aumente o número de módulos operacionais.

7.3 Características da Ferramenta

O CSAR foi criado utilizando duas linguagens de programação: C e PROLOG. Conforme as características das rotinas a serem desenvolvidas, foi escolhida uma ou outra linguagem. Algoritmos que exigiam velocidade de execução foram desenvolvidos na linguagem C. Já os algoritmos que exigiam muitas consultas e modificaçöes em estruturas ou bancos de dados foram escritos em PROLOG.

A programação foi dividida nos seguintes módulos:

i- Rotina de compilação da linguagem de descrição OPCODE para formato intermediário composto de fatos da linguagem PROLOG (em linguagem C).

ii- Rotina de interpretação do formato intermediário composto de fatos PROLOG e divisão da arquitetura em parte operativa e parte de controle (em linguagem PROLOG).

iii- Rotina de posicionamento dos elementos da parte operativa (em linguagem C).

iv- Rotina de definição do número total de barramentos a serem utilizados e qual conectores são ligados a cada barramento (em linguagem PROLOG).

v- Rotina de expansão simbólica (em linguagem C).

vi- Rotina de montagem de módulos (em linguagem C).

vii- Rotina de avaliação de desempenho (em linguagem PROLOG).

viii- Rotina de tratamento de irregularidades (em linguagem PROLOG).

As rotinas utilizam como interface arquivos intermediários. A rotina i, por exemplo, gera um arquivo intermediário que é lido pela rotina ii. Os arquivos são mantidos mesmo após a compilação, permitindo ao usuário uma maior avaliação dos resultados obtidos para diferentes descrições do mesmo circuito.

8. AVALIAÇÃO DA FERRAMENTA

8.1 Exemplos de Utilização

Para avaliar e compreender melhor todo o processo de compilação do CSAR desde a linguagem de descrição OPCODE, dois exemplos são estudados. Apesar de os exemplos serem pequenos (poucas operaçoes) envolvem as principais rotinas do CSAR.

8.1.1 Divisão Lenta

Este exemplo mostra a implementação de um algoritmo de divisão, utilizando módulos operacionais deslocadores e somadores.

O algoritmo de Divisão Lenta realiza a seguinte operação:

Divisão do registrador A pelo registrador B. O resto da divisão é colocado no registrador R e o quociente da divisão no registrador Q.

O algoritmo pode ser descrito em uma linguagem do tipo PASCAL da seguinte forma:

```
{ Divisão A/B - R:resto, Q:quociente }
C := B;
while C <= A do C := C * 2;
R := A;
Q := 0;
```

```
while C > B do
     begin
     Q := Q * 2;
     C := C / 2;
     if R >= C then
          begin
          Q := Q + 1;
          R := R - C;
          end;
     end;
end;
```

A mesma descrição pode ser feita utilizando a linguagem OPCODE:

```
/* Declaração dos Registradores Utilizados */
Palavra   A 8 e, /* entrada 1    */
          B 8 e, /* entrada 2    */
          C 8 t, /* temporário   */
          R 8 s, /* resto        */
          Q 8 s, /* quociente    */
          1 8 c, /* constante 1 */
          0 8 c; /* constante 0 */

BitSlices 8; /* Número de Bit Slices a ser utilizado */

#1 : C <- B ;
#2 : (#3) <- C <= A / #4 ;
#3 : C <- << C / #2 ;
```

```
#4 : R <- A / Q <- 0 ( phi 2 ) / (#5) <- C > B / #8 ;
#5 : Q <- << Q / C <- >> C ;
#6 : (#7) <- R >= C / #9 ;
#7 : Q <- Q + 1 / R <- R - C / (#5) <- C > B / #1 ;
#8 : (#5) <- C > B / #1 ;
```

Após a compilação da linguagem de entrada o CSAR separa as informações que interessam à parte operativa e as informações que interessam à parte de controle.

Parte Operativa

Como não existe ainda nenhum algoritmo de otimização do uso de registradores no CSAR, os registradores declarados são automaticamente implementados.

Já os módulos operacionais são otimizados de modo a evitar a repetição de módulos operacionais não necessários. O resultado da compilação é:

```
CSAR *** Registradores:

R1 ::: A <0:7>
R2 ::: B <0:7>
R3 ::: C <0:7>
R4 ::: R <0:7>
R5 ::: Q <0:7>
R6 ::: 1 <0:7>
R7 ::: 0 <0:7>
```

```
CSAR ***  Módulos Operacionais:

     MO1 ::: Comparador              ( )   n. 1
     MO2 ::: Deslocador Esquerda     <<    n. 1
     MO3 ::: Deslocador Direita      >>    n. 1
     MO4 ::: Comparador              ( )   n. 2
     MO5 ::: Somador                 +     n. 1
     MO6 ::: Subtrator               -     n. 1
```

Junto com a descrição dos registradores e módulos operacionais, o CSAR apresenta uma relação das conexoes a serem realizadas em cada ciclo da máquina de controle.

```
CSAR *** Conectores

     A_e1       ::: Registrador A - Entrada
     A_s1       ::: Registrador A - Saída
     B_e1       ::: Registrador B - Entrada
     B_s1       ::: Registrador B - Saída
     C_e1       ::: Registrador C - Entrada
     C_s1       ::: Registrador C - Saída
     R_e1       ::: Registrador R - Entrada
     R_s1       ::: Registrador R - Saída
     Q_e1       ::: Registrador Q - Entrada
     Q_s1       ::: Registrador Q - Saída
     1_s1       ::: Registrador 1 - Saída
     0_s1       ::: Registrador 0 - Saída
     MO1_e1     ::: Mod.Operac. 1 - Entrada.
     MO1_e2     ::: Mod.Operac. 1 - Entrada.
```

```
M02_e1     ::: Mod.Operac. 2 - Entrada.
M02_s1     ::: Mod.Operac. 2 - Saida.
M03_e1     ::: Mod.Operac. 3 - Entrada.
M03_s1     ::: Mod.Operac. 3 - Saida.
M04_e1     ::: Mod.Operac. 4 - Entrada.
M04_e2     ::: Mod.Operac. 4 - Entrada.
M05_e1     ::: Mod.Operac. 5 - Entrada.
M05_e2     ::: Mod.Operac. 5 - Entrada.
M05_s1     ::: Mod.Operac. 5 - Saida.
M06_e1     ::: Mod.Operac. 6 - Entrada.
M06_e2     ::: Mod.Operac. 6 - Entrada.
M06_s1     ::: Mod.Operac. 6 - Saida.
```

CSAR *** Conexões

```
Ciclo 1 :::  estado 000 ::: Fase 1
C_e1 <---> B_s1

Ciclo 1 :::  estado 000 ::: Fase 2
---

Ciclo 2 :::  estado 001 ::: Fase 1
M01_e1 <---> C_s1
M01_e2 <---> A_s1

Ciclo 2 :::  estado 001 ::: Fase 2
---

Ciclo 3 :::  estado 010 ::: Fase 1
M02_e1 <---> C_s1
```

```
Ciclo 3 :::  estado 010 ::: Fase 2
M02_s1 <---> C_e1

Ciclo 4 :::  estado 011 ::: Fase 1
R_e1 <---> A_s1
M04_e1 <---> C_s1
M04_e2 <---> B_s1

Ciclo 4 :::  estado 011 ::: Fase 2
Q_e1 <---> 0_s1

Ciclo 5 :::  estado 100 ::: Fase 1
M02_e1 <---> Q_s1
M03_e1 <---> C_s1

Ciclo 5 :::  estado 100 ::: Fase 2
M02_s1 <---> Q_e1
M03_s1 <---> C_e1

Ciclo 6 :::  estado 101 ::: Fase 1
M01_e1 <---> R_s1
M01_e2 <---> C_s1
M04_e1 <---> C_s1
M04_e2 <---> B_s1

Ciclo 6 :::  estado 101 ::: Fase 2
---

Ciclo 7 :::  estado 110 ::: Fase 1
M05_e1 <---> Q_s1
M05_e2 <---> 1_s1
```

```
M06_e1 <---> R_s1
M06_e2 <---> C_s1
M04_e1 <---> C_s1
M04_e2 <---> B_s1

Ciclo 7 :::  estado 110 ::: Fase 2
M05_s1 <---> Q_e1
M06_s1 <---> R_e1

Ciclo 8 :::  estado 111 ::: Fase 1
M04_e1 <---> C_s1
M04_e2 <---> B_s1

Ciclo 8 :::  estado 111 ::: Fase 2
---
```

Conhecendo os elementos que fazem parte do bit-slice, bem como as conexões existentes entre estes elementos, o CSAR tem informações suficientes para realizar o posicionamento.

O critério escolhido para posicionamento foi uma função custo com k1=0.2, k2=0.2, k3=0.4, k4=0.2, buscando um equilíbiro entre área e velocidade final.

O resultado do posicionamento, bem como as rotas que devem existir para as conexões podem ser vistos na fig. 8.1.

Para o posicionamento encontrado pelo CSAR, é necessário apenas um barramento e quatro chaves de barramento.

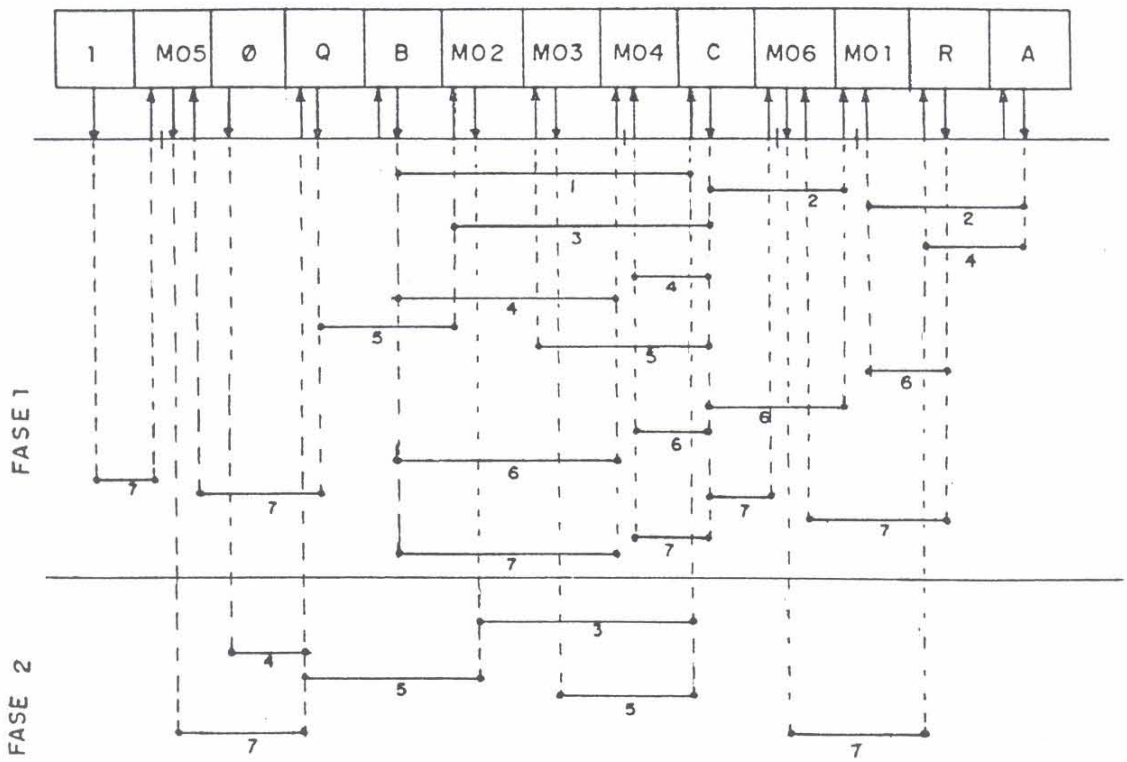


Figura 8.1 Slow-Division Posicionamento e Roteamento

CSAR ::: Resultados do Posicionamento e Roteamento

Critério :

k1=0.2

k2=0.2

k3=0.4

k4=0.2

tempo: 622s

Posicionamento:

OO ::: RG -> 1

O1 ::: MO -> M05

```
02 ::: RG -> 0
03 ::: RG -> Q
04 ::: RG -> B
05 ::: MO -> MO2
06 ::: MO -> MO3
07 ::: MO -> MO4
08 ::: RG -> C
09 ::: MO -> MO6
10 ::: MO -> MO1
11 ::: RG -> R
12 ::: RG -> A
```

Total de Barramentos : 1
Total de Chaves de Barramento: 4
Descriçåo:
Chave 1 : Barramento 1 : Entre Conectores MO5_e1 e MO5_s1
Chave 2 : Barramento 1 : Entre Conectores MO4_e1 e MO4_e2
Chave 3 : Barramento 1 : Entre Conectores MO6_e1 e MO6_s1
Chave 4 : Barramento 1 : Entre Conectores MO1_e1 e MO1_e2

Parte de Controle

As equaçöes do PLA devem existir para todas as chaves presentes na arquitetura da parte operativa.

CSAR ::: Parte de Controle

-Número de Estados : 8 --> 3 variáveis (e2, e1, e0)
-Equaçöes: Chave = e2 e1 e0 (fase)

Controle dos Conectores Ligados aos Barramentos

```
A_e1 = ?
A_s1 = 001(1) + 011(1)
B_e1 = ?
B_s1 = 000(1) + 011(1) + 101(1) + 110(1)
C_e1 = 000(1) + 010(2) + 100(2)
C_s1 = 001(1) + 010(1) + 011(1) + 100(1) + 101(1) + 110(1)
R_e1 = 011(1) + 110(2)
R_s1 = 101(1) + 110(1)
Q_e1 = 011(2) + 100(2) + 110(2)
Q_s1 = 100(1) + 110(1)
1_s1 = 110(1)
0_s1 = 011(2)
M01_e1 = 001(1) + 101(1)
M01_e2 = 001(1) + 101(1)
M02_e1 = 010(1) + 100(1)
M02_s1 = 010(2) + 100(2)
M03_e1 = 101(1)
M03_s1 = 100(2)
M04_e1 = 011(1) + 101(1) + 110(1) + 111(1)
M04_e2 = 011(1) + 101(1) + 110(1) + 111(1)
M05_e1 = 110(1)
M05_e2 = 110(1)
M05_s1 = 110(2)
M06_e1 = 110(1)
M06_e2 = 110(1)
M06_s1 = 110(2)
```

Controle das Chaves de Barramento

Chave 1 = 110(1)
Chave 2 = 011(1) + 101(1) + 110(1) + 111(1)
Chave 3 = 110(1)
Chave 4 = 101(1) + 001(1)

Controle da Máquina de Estados

	Próx. Estado	
Estado Atual	Condição Verdadeira	Condição Falsa
1	2	2
2	3	4
3	2	2
4	5	8
5	6	6
6	7	1
7	5	1
8	5	1

8.1.2 Multiplicação

Este segundo exemplo mostra a implementação de um algoritmo de multiplicação, também utilizando apenas módulos operacionais deslocadores e somadores como base.

O algoritmo de Multiplicação realiza a seguinte operação: multiplicação do registrador M pelo registrador N. O resultado da multiplicação é colocado no registrador R. O algoritmo pode ser descrito em uma linguagem do tipo PASCAL

da seguinte forma:

```
{ Multiplicação M*N - R:resultado }
begin
B := 1;
R := 0;
while B <> 0 do
     begin
     A := B and M;
     if ( A <> 0 ) then R := R + N;
     N := N * 2;
     B := B * 2;
     end;
end.
```

A mesma descrição pode ser feita utilizando a linguagem OPCODE:

```
/* Declaração dos Registradores Utilizados */
Palavra   A 8 e, /* entrada 1    */
          B 8 e, /* entrada 2    */
          M 8 t, /* temporário   */
          N 8 t, /* temporário   */
          R 16 s, /* resultado   */
          1 8 c, /* constante 1 */
          0 8 c; /* constante 0 */

BitSlices 8; /* Número de Bit Slices a ser utilizado */
```

```
#1: M <- M / N <- N / B <- 1 / R <- 0;
#2: (#3) <- B <> 0 / #7;
#3: A <- B & M;
#4: (#5) <- A <> 0 / #6;
#5: R <- R + N;
#6: N <- << N / B <- << B / #2;
#7: R <- R / #1;
```

Parte Operativa

O resultado da compilação é:

```
CSAR *** Registradores:

     R1 ::: M <0:7>
     R2 ::: N <0:7>
     R3 ::: R <0:15>
     R4 ::: B <0:7>
     R5 ::: A <0:7>
     R6 ::: 1 <0:7>
     R7 ::: 0 <0:7>

CSAR ***  Módulos Operacionais:

     M01 ::: Comparador            ( )   n. 1
     M02 ::: And                   &     n. 1
     M03 ::: Somador               +     n. 1
     M04 ::: Somador               +     n. 2
     M05 ::: Deslocador Esquerda   <<    n. 1
     M06 ::: Deslocador Esquerda   <<    n. 2
```

CSAR *** Conectores

```
M_e1       ::: Registrador M - Entrada
M_s1       ::: Registrador M - Saída
N_e1       ::: Registrador N - Entrada
N_s1       ::: Registrador N - Saída
R_e1       ::: Registrador R - Entrada
R_e2       ::: Registrador R - Entrada
R_s1       ::: Registrador R - Saída
R_s2       ::: Registrador R - Saída
B_e1       ::: Registrador B - Entrada
B_s1       ::: Registrador B - Saída
A_e1       ::: Registrador A - Entrada
A_s1       ::: Registrador A - Saída
1_s1       ::: Registrador 1 - Saída
0_s1       ::: Registrador 0 - Saída
MO1_e1     ::: Mod.Operac. 1 - Entrada.
MO1_e2     ::: Mod.Operac. 1 - Entrada.
MO2_e1     ::: Mod.Operac. 2 - Entrada.
MO2_e2     ::: Mod.Operac. 2 - Entrada.
MO2_s1     ::: Mod.Operac. 2 - Saída.
MO3_e1     ::: Mod.Operac. 3 - Entrada.
MO3_e2     ::: Mod.Operac. 3 - Entrada.
MO3_s1     ::: Mod.Operac. 3 - Saída.
MO4_e1     ::: Mod.Operac. 4 - Entrada.
MO4_e2     ::: Mod.Operac. 4 - Entrada.
MO4_s1     ::: Mod.Operac. 4 - Saída.
MO5_e1     ::: Mod.Operac. 5 - Entrada.
MO5_s1     ::: Mod.Operac. 5 - Saída.
```

M06_e1 ::: Mod.Operac. 6 - Entrada.
M06_s1 ::: Mod.Operac. 6 - Saída.

Note-se que o registrador R de 16 bits foi dividido em dois registradores de 8 bits.

CSAR *** Conexões

Ciclo 1 ::: estado 000 ::: Fase 1
1_s1 <---> B_e1
0_s1 <---> R_e1
0_s1 <---> R_e2

Ciclo 1 ::: estado 000 ::: Fase 2

Ciclo 2 ::: estado 001 ::: Fase 1
B_s1 <---> M01_e1
0_s1 <---> M01_e2

Ciclo 2 ::: estado 001 ::: Fase 2

Ciclo 3 ::: estado 010 ::: Fase 1
B_s1 <---> M02_e1
M_s1 <---> M02_e2

Ciclo 3 ::: estado 010 ::: Fase 2
M02_s1 <---> A_e1

Ciclo 4 ::: estado 011 ::: Fase 1

```
A_s1 <---> M01_e1
O_s1 <---> M01_e2

Ciclo 4 :::  estado 011 ::: Fase 2
---

Ciclo 5 :::  estado 100 ::: Fase 1
R_s1 <---> M03_e1
R_s2 <---> M04_e1
N_s1 <---> M04_e2

Ciclo 5 :::  estado 100 ::: Fase 2
M03_s1 <---> R_e1
M04_s1 <---> R_e2

Ciclo 6 :::  estado 101 ::: Fase 1
N_s1 <---> M05_e1
B_s1 <---> M06_e1

Ciclo 6 :::  estado 101 ::: Fase 2
M05_s1 <---> N_e1
M06_s1 <---> B_e1

Ciclo 7 :::  estado 110 ::: Fase 1
---

Ciclo 7 :::  estado 110 ::: Fase 2
---

Ciclo 8 :::  estado 111 ::: Fase 1
```

Ciclo 8 ::: estado 111 ::: Fase 2

O critério escolhido para posicionamento foi uma função custo com k1=0.2, k2=0.2, k3=0.4, k4=0.2, buscando um equilíbiro entre área e velocidade final.

O resultado do posicionamento, bem como as rotas que devem existir para as conexões pode ser visto na fig. 8.2.

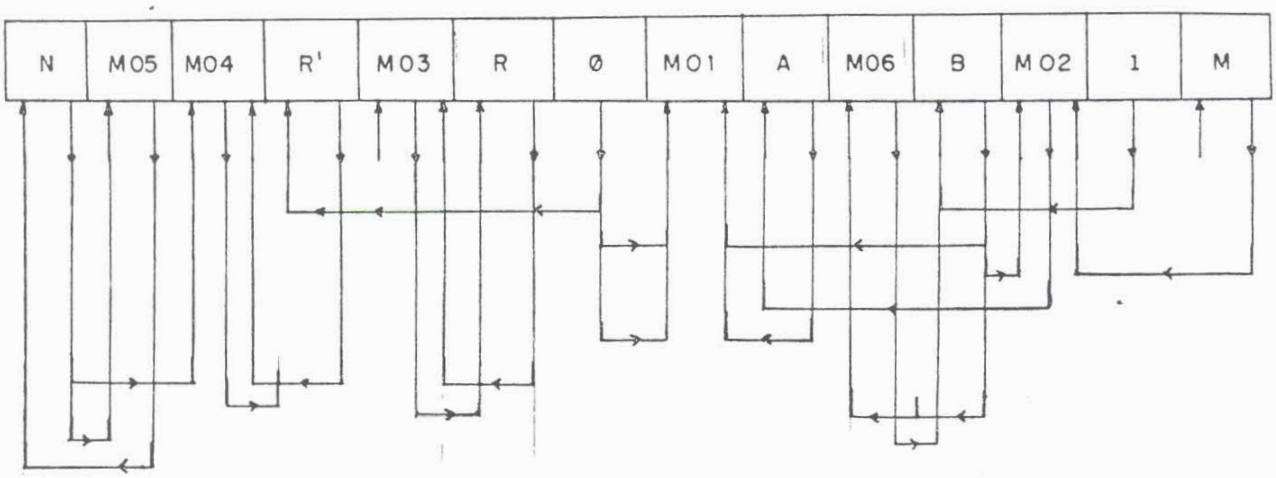


Figura 8.2 Multiplicação: Posicionamento e Roteamento

Para o posicionamento encontrado pelo CSAR, é necessário apenas um barramento e quatro chaves de barramento.

```
CSAR ::: Resultados do Posicionamento e Roteamento

Critério :
k1=0.2
k2=0.2
k3=0.4
k4=0.2
tempo: 529s

Posicionamento:
     00 ::: RG -> N
     01 ::: MO -> MO5
     02 ::: MO -> MO4
     03 ::: RG -> R
     04 ::: MO -> MO3
     05 ::: RG -> R'
     06 ::: RG -> O
     07 ::: MO -> MO1
     08 ::: RG -> A
     09 ::: MO -> MO6
     10 ::: RG -> B
     11 ::: MO -> MO2
     12 ::: RG -> 1
     13 ::: RG -> M


Total de Barramentos : 1
Total de Chaves de Barramento: 4
```

Descriçåo:

Chave 1 : Barramento 1 : Entre Conectores M04_e1 e M04_s1

Chave 2 : Barramento 1 : Entre Conectores M03_e1 e M03_e2

Chave 3 : Barramento 1 : Entre Conectores M01_e1 e M01_e2

Chave 4 : Barramento 1 : Entre Conectores M02_s1 e M02_e2

Parte de Controle

CSAR ::: Parte de Controle

-Número de Estados : 8 --> 3 variáveis (e2, e1, e0)

-Equaçöes: Chave = e2 e1 e0 (fase)

Controle dos Conectores Ligados aos Barramentos

M_e1 = ?

M_s1 = 010(1)

N_e1 = 101(2)

N_s1 = 100(1) + 101(1)

R_e1 = 000(1) + 100(2)

R_e2 = 000(1) + 100(2)

R_s1 = 100(1)

R_s2 = 100(1)

B_e1 = 000(1) + 101(2)

B_s1 = 001(1) + 010(1) + 101(1)

A_e1 = 011(1) + 010(2)

A_s1 = ?

0_s1 = 000(1) + 001(1) + 011(1)

1_s1 = 000(1)

M01_e1 = 001(1) + 011(1)

M01_e2 = 001(1) + 011(1)

M02_e1 = 010(1)
M02_e2 = 010(1)
M02_s1 = 010(2)
M03_e1 = 100(1)
M03_e2 = 100(1)
M03_s1 = 100(2)
M04_e1 = 100(1)
M04_e2 = 100(1)
M04_s1 = 100(2)
M05_e1 = 101(1)
M05_s1 = 101(2)
M06_e1 = 101(1)
M06_s1 = 101(2)

Controle das Chaves de Barramento

Chave 1 = 100(1)
Chave 2 = 100(2)
Chave 3 = 000(1) + 001(1) + 100(1)
Chave 4 = 010(1)

Controle da Máquina de Estados

	Próx. Estado	
Estado Atual	Condição Verdadeira	Condição Falsa
1	2	2
2	3	7
3	4	4
4	5	6
5	6	6
6	2	2
7	1	1

8	1	1

8.1.2 Conclusões

Deve-se destacar nos dois exemplos o número final de barramentos (1) para os dois exemplos. As chaves de barramento (4) não chegam a comprometer a parte de controle, reforçando a metodologia adotada.

Apesar de que o paralelismo de operações para os dois exemplos seja pequeno (3 a 4 operações), a metodologia adotada pode suportar um aumento de paralelismo de operações sem comprometer a área e velocidade do circuito final.

9. CONCLUSÃO

Conseguiu-se realizar uma ferramenta de CAD que gera uma arquitetura alvo correspondente aos objetivos deste trabalho, ou seja, voltada à execução de operações paralelas. Procurou-se detalhar todos os passos envolvidos na geração da arquitetura, tais como:

- Escolha de uma linguagem de entrada: neste caso optou-se por criar uma linguagem que descrevesse as operações baseando-se nas estruturas de controle e operacionais que deveriam existir na arquitetura alvo.

- Divisão da arquitetura em parte operativa e parte de controle; escolha das estruturas da parte operativa (barramentos, registradores, operadores, etc.) e parte de controle (PLA).

- Algoritmos para escolha do número de barramentos e chaves de barramento, posicionamento dos registradores e operadores da parte operativa conforme diversos critérios (associados aos compromissos de velocidade e área do circuito) e roteamento.

- Parametrização de células visando independência de tecnologia.

- Resolução de irregularidades na arquitetura.

Analisando-se os exemplos apresentados (divisor e multiplicador) pode-se perceber duas características básicas encontradas durante o processo de compilação:

- A compilação de um nível de descrição para outro pode ser simplificado adotando-se um algoritmo específico para realizar esta atividade.

- A escolha do algoritmo e de que fatos devem ser abstraídos durante as passagens de nível é que dizem se o compilador gera uma arquitetura aproveitável ou não.

Que o CSAR parte de uma linguagem de descrição de alto nível e gera uma arquitetura alvo no nível de layout não há dúvidas. A questão é: o layout gerado é aproveitável, satisfazendo as restrições de área, consumo e velocidades exigidas pelo usuário ou, ao menos, comparável com circuitos ASIC?

A avaliação dos resultados leva as seguintes conclusões:

- A divisão da arquitetura em parte operativa e parte de controle ajuda na divisão e solução do problema.

- A parte operativa apresenta o posicionamento e o roteamento automático, prevendo soluções combinatórias não previstas pelo usuário. Pode-se dizer o mesmo da parte de controle e dos geradores de módulos em geral. Entretanto todas as vantagens da automatização desta fase podem ser perdidas se a especificação do usuário não for otimizada por algorítmos de síntese de alto nível, pois, como já foi discutido, o "gargalo" do sistema está nos níveis mais altos de descrição.

- A parametrização de células ainda não atinge resultados próximos aos projetos manuais. Porém a utilização de compactadores de células torna a solução bastante viável.

O CSAR pode gerar soluções eficientes comparadas às soluções manuais. As perdas em área podem ser comparadas pela velocidade e confiabilidade de resolução do problema. Entretanto a dependência e iteração com o usuário deverá continuar existindo, exigindo conhecimentos específicos em microeletrônica por parte do usuário.

A ferramenta deve ser melhorada nos seguintes pontos (que servem como sugestão e perspectivas de continuação do trabalho):

- Aproveitamento de outras estruturas de controle,
- criação de uma interface mais "amigável" ao usuário,
- linguagem de entrada mais próxima ao usuário (síntese de alto nível),
- aumento de módulos operacionais,
- utilização de outros parametrizadores de células e compactadores.

A conclusão final deste trabalho é de que os compiladores de silício são ferramentas viáveis e que fazem parte de uma realimentação positiva de processo de fabricação de circuitos integrados: construir circuitos integrados melhores, computadores melhores, ferramentas de CAD melhores, e, novamente, circuitos integrados melhores.

BIBLIOGRAFIA

[AKI79] AKINO, S. et al. "Circuit Simulation and Timing Verification Based on MOS/LSI Mask Information", Proceedings of the 16th Design Automation Conference, pp. 88-94, 1979.

[AYR79a] AYRES, R. "IC Specification Language". Proceeedings of the 16th Design Automation Conference, pp 307-309, 1979.

[AYR79b] AYRES, R. "Silicon Compilation: A Hierarchical Use of PLAs". Proceedings of the 16th Design Automation Conference, pp 314-326, 1979.

[BAR77] BARBACCI, M. "The ISPS Computer Description Language", Carnegie-Mellon University Technical Report, 1977.

[BAR81] BARBACCI, M. "ISPS: The Notation and Specifications", IEEE Transactions on Computers, Vol.C-30, pp. 24-40, jan 1981.

[BLA85] BLACKMAN, T. et al. "The SILC Silicon Compiler: Languages and Features", Proceedings of the 22nd Design Automation Conference, 1985.

[BOY88]BOYER, D. "Symbolic Layout Compaction Review", Proceedings of the 25th ACM/IEEE Design Automation Conference, pp. 383-89, 1988.

[BRA84] BRAYTON, R. et al. "VLSI Computer Architecture and Digital Signal Processing: Logic Minimisation Algorithms for VLSI Synthesis", Hingham, MA: Kluwer Academic Publ., 1984.

[BRY86] BRYAN, T. ; KARGER, P. "Automatic Placement: A Review of Current Techniques", Proceedings of the 23rd Design Automation Conference, pp. 622-629, 1986.

[BUS86] BUSHNELL, M. ; DIRECTOR, S. "VLSI CAD Tool Integration Using the ULYSSES Environment", Proceedings of the 23rd Design Automation Conference, pp. 55-61, june 1986.

[CAR87] CARRO, L. ; SUZIM, A. "Metodologia em Concepção de Circuitos Integrados", 1[Congresso da Sociedade Brasileira de Microeletrônica, p. 243, 1987.

[CAR89] CARRO, L. et al. "Gerador Parametrizável de Partes Operativas CMOS", IV Congresso da Sociedade Brasileira de Microeletrônica, pp. 47-58, 1989.

[CES87] CESEAR, T. et al. "PAMS: An Expert System for Parameterized Module Synthesis", Proceedings of the 24th Design Automation Conference, pp. 666-671, 1987.

[CHE77] CHEN, K. "The Chip Layout Problem: An Automatic Wiring Procedure", Proceedings of the 14th Design Automation Conference, pp. 298-302, 1977.

[CHO85] CHO, Y. "A Subjective Review of Compactation", Proceedings of the ACM/IEEE 22nd Design Automation Conference, pp. 396-404, June 1985.

[CIA75] CIAMPI, P. "A System for Solution of the Placement Problem", Proceedings of the 12th Design Automation Conference, pp. 406-413, 1975.

[CHU82] CHUQUILLANQUI, S. ; PEREZ, T. "PAOLA: A Tool for Topological Optimization of Large PLAs", Proceedings of the 19th Design Automation Conference, 1982.

[COR79] CORRIGAN, L. "A Placement Capability Based on Partitioning", Proceedings of the 16th Design Automation Conference, pp. 406-413, 1979.

[COR81] CORY, W. "Symbolic Simulation for Functional Verification with ADLIB and SDL", Proceedings of the 18th Design Automation Conference, pp. 82-89, 1981.

[COT80] COTE, L. "The Interchange Algorithms for Circuit Placement Problems", Proceedings of the 17th Design Automation Conference, pp. 451-457, 1980.

[COX80] COX, G. "The Standard Transistor Array part II: Automatic Cell Placement Techniques", Proceedings of the 17th Design Automation Conference, pp. 451-457, 1980.

[DEV86] DEVADAS, S. "GENIE: A Generalized Array Optimizer for VLSI Synthesis", Proceedings of the 23rd Design Automation Conference, June 1986.

[DIL89] DILLENBURG, R. "Gerador de PLA com Dimensionamento de Transistores", IV Congresso da Sociedade Brasileira de Microeletrônica, pp. 59-68, 1989.

[DUN80] DUNLOP, A. "SLIM: The Translation of Symbolic Layouts into Mask Data", Proceedings of the 17th Design Automation Conference, pp. 595-602, June 1980.

[EDW80] EDWARD, M. et al. "Design Integrity and Imunity Checking", Proceedings of the 17th Design Automation Conference, 1980.

[FEL76] FELLER, A. "Automatic Layout of Low-Cost Quick-Turnaround Random-Logic Custom LSI Devices", Proceedings of the 13th Design Automation Conference, pp. 79-85, 1976.

[FOS80] FOSTER, M. "The Design of Special Purpose VLSI Chips", IEEE Computer, pp. 26-40, jan 1980.

[FRA89] FRANC, B. "Automated Synthesis for Testability", IV Congresso da Sociedade Brasileira de Microeletrônica, pp. 169-184, 1989.

[GAJ84] GAJSKY, D. "Silicon Compilers and Expert Systems for VLSI", Proceedings of the 21st Design Automation Conference, p.86-87, 1984.

[GAJ85] GAJSKY, D. "Silicon Compilation", VLSI System Design, Nov 1985.

[GAJ86] GAJSKY, D. et al. "Silicon Compilation (Tutorial)", IEEE Custom Integrated Circuits Conference, pp. 102-110, 1986.

[GOE87] GOERING, R. "Silicon Compilers bridge gap between concepts and silicon", Computer Design, pp. 67-80, November, 1987.

[GOL85] GOLBERG, A. et al. "Approaches Toward Silicon Compilation", IEEE Circuits and Devices, Vol.1, No.3, May 1985.

[GOM88] GOMES, R. "DARC: Um Verificador de Regras de Projeto de CIs Utilizando Programação em Lógica", III Simpósio Brasileiro para Concepção de Circuitos Integrados, pp. 85-94, Abril 1988.

[GRA79] GRAY, J. "Introduction to Silicon Compilation", Proceedings of the 16th Design Automation Conference, pp. 305-306, 1979.

[GRA82] GRAY, J. "Designing Gate Arrays Using a Silicon Compiler". Proceedings of the 19th Design Automation Conference, pp. 377-383, 1982.

[GRO83] GROSS,R. "Silicon Compilers: A Critical Survey". Department of Computer Science, University of North Carolina at Chapel Hill, may 1983.

[HAF82] HAFER, L. "Automated Synthesis of Digital Hardware". IEEE Transactions on Computers, pp. 20-28, feb 1982.

[HED87] HEDENSTIERNA, N. ; JEPPSON, K. "New Algorithms for Increased Efficiency in Hierarquical Design Rule Checking", INTEGRATION, the VLSI journal, pp. 319-336, May 1987.

[HIR86] HIRAYAMA, M. "A Silicon Compiler System Based on Asynchronous Architecture", Proceedings of the IEEE International Conference on CAD, 1985.

[HIT83] HITCHCOCK, C. ; THOMAS, D., "A Method of Automatic Data Path Synthesis", Proceedings of the 20th Design Automation Conference, pp. 484-489, Jun 1983.

[JAC86] JACOBI, R. "Microeditor: Editor Gráfico para Microeletrônica em PC". Congresso da Sociedade Brasileira de Microeletrônica, pp. 408-418, Jul 1986.

[JAC88] JACOBI, R. et al. "EMHIR: Editor de Máscaras Hierárquico para Layout VLSI", III Simpósio Brasileiro para Concepção Circuitos Integrados, pp. 1-7, Abril 1988.

[JAM85] JAMIER, R. "APOLLON: A Data-Path Silicon Compiler", IEEE Circuits and Devices, Vol.3, May 1985.

[JHO85] JHON, C. et al. "Silicon Compilation Based on a Data-Flow Paradigm",IEEE Circuits and Devices, Vol.1, No.3, May 1985.

[JEN82] JENNE, D. ; STAMM, D. "Managing VLSI Complexity: A User's Viewpoint" VLSI Design, pp. 14-20, march 1982.

[JER86] JERRAYA, A. et al. "Principles of the SYCO Compiler", Proceedings of the 23rd Design Automation Conference, pp. 715-721, 1986.

[JOH79] JOHANNSEN, D. "Bristle Blocks: A Silicon Compiler". Proceedings of the 16th Design Automation Conference, pp. 310-313, 1979.

[KAM79] KAMBAYASHI, Y. "Logic Design of Programmable Logic Arrays", IEEE Transactions on Computers, pp. 609-617, september 1979.

[KAW82] KAWATO, N. et al. "An Interactive Logic Synthesis System Based Upon AI Techniques". Proceedings of the 19th Design Automation Conference, pp. 858-864, 1982.

[LIN76] LINDSAY, B. ; PREAS, B. "Design Rule Checking and Analysis of CI Mask Designs", Proceedings of the 13th Design Automation Conference, pp. 301-308, June 1976.

[LUB87] LUBASZEWSKI et al. "Concepção de um CI para Controle Industrial", VII Congresso da SBC, Jul 1987.

[LUB89] LUBASZEWSKI et al. "A Random Logic Generator Using the TRANCA Methodology", IV Congresso da Sociedade Brasileira de Microeletrônica, pp. 69-80, 1989.

[LUK82] LUKUHAY, J. ; KUBITZ, W. "A Layout Synthesis System for nMOS Gate-Cells", Proceedings of the 19th Design Automation Conference, pp. 307-314, 1982.

[MAR86] MARSHBURN, I. et al. "DATAPATH: A Cmos Data Path Silicon Assembler". Proceedings of the 23rd Design Automation Conference, pp. 722-729, 1986.

[MCC79] McCAW, C. "Unified Shapes Checker: A Checking Tool for LSI", Proceedings of the 16th Design Automation Conference, pp. 81-87, June 1979.

[MCC84] McCLUSKEY, E. "A Survey of Design for Testability Scan Techniques", VLSI Design, Vol.V, No.12, Dec 1984.

[MCF86] McFARLAND, M. "Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions", Proceedings of the 23rd Design Automation Conference, pp. 474-480, 1986.

[MEA80] MEAD, C. ; CONWAY, L. "Introduction to VLSI System", Reading, MA: Addison-Wesley, Inc. 1980.

[MOR89] MORAES, F. ; REIS, R. "Sintese Automática de Células Utilizando Estratégia Gate-Matrix", IV Congresso da Sociedade Brasileira de Microeletrônica, pp. 231-242, 1989.

[NAG82] NAGLE, A. et al. "Synthesis of Hardware for the Control of Digital Systems", IEEE Transactions on Computer Aided Design, vol. CAD-1, No.4, pp. 201-212, 1982.

[NEW87] NEWTON, A. et al. "CAD Tools for ASIC Design", Proceedings of the IEEE, Vol.75, No.6, June 1987.

[NIE83] NIESSEN, C. "Hierarchical Design Methodologies and Tools for VLSI Chips". Proceedings of the IEEE 71, pp.66-75, sep 1979.

[OLI88] OLIVEIRA, F. "Uma Proposta de Sistema Especialista para Floor-Planning de circuitos Standard-Cell", III Simpósio Brasileiro de Concepção de Circuitos Integrados, pp. 75-83, Abril 1988.

[OST79] OSTAPKO, D. ; HONG, S. "Fault Analysis and Test Generation for Programmable Logic Arrays", IEEE Transactions on Computers, pp. 617-627, sep 1979.

[PAR85] PARK, N. "Synthesis of Optimal Clocking Schemes", Proceedings of the 22nd Design Automation Conference, 1985.

[PAR86a] PARKER,A. ; PARK, N. "SEHWA: A Program for Synthesis of Pipelines", Proceedings of the 23rd Design Automation Conference, pp. 454-460, june 1986.

[PAR86b] PARKER,A. et al. "MAHA: A Program for Datapath Synthesis", Proceedings of the 23rd Design Automation Conference, pp. 461-466, june 1986.

[PAR87] PARKER,A. ; HAYATI,S. "Automating the VLSI Design Process Using Expert Systems and Silicon Compilation". Proceedings of the IEEE, pp. 777-785, june 1987.

[PAT79] PATIL, S. ; WELCH, T. "A Programmable Logic Approach for VLSI", IEEE Transactions on Computers, pp. 594-601, sep 1979.

[REI83] REIS, R. "TESS: Evaluateur Topologique Predictif pour la Generation Automatique des Plans de Masse de Circuits VLSI" Tese (Doct.Ing.) Institut National Polytechnique de Grenoble, Grenoble, Franca, 1983.

[REI83] REIS, R. "Uma Metodologia Descendente de Concepcão de Circuitos Integrados", Simpósio Brasileiro de Concepcão de Circuitos Integrados, 1983.

[REI86] REIS, R. et al. "Design of the SYCO 6502 using the SYCO compiler", International Conference on Computer Design, 1986.

[REI87a] REIS, R. "A Microeletrônica na UFRGS", 20 Congresso Nacional de Informática, 1987.

[REI87b] REIS, R. "A New Standard Cell CAD Methodology", IEEE Custom Integrated Circuits Conference, Portland, EUA, Maio 1987.

[REI88] REIS, R. et al. "An Efficient Design Methodology for Standard Cell Circuits", IEEE International Simposyum on Circuits and Systems, Helsinki, Finlândia, Junho 1988.

[SEQ83] SEQUIN, C. "Managing VLSI Complexity: An Outlook". Proceedings of the IEEE, pp. 149-166, Jan 1983.

[SHI80] SHIU, C. "A Hierarchical Approach for Layout Versus Circuit Consistency Check", Proceedings of the 17th Design Automation Conference, pp. 269-276, 1980.

[SHI83]SHIVA, S. "Automatic Hardware Synthesis". Proceedings of the IEEE, pp. 76-87, Jan 1983.

[SIL89] SILVA, J. "Otimização de Variaveis no Processo de Síntese Automática", IV Congresso da Sociedade Brasileira de Microeletrônica, pp. 243-254, 1989.

[SMI82] SMITH, K. et al. "Structured Logic Design of Integrated Circuits Using the Storage Logic Array", IEEE Journal of Solid State Circuits, pp. 43-52, 1982.

[SOU83] SOUTHARD, J. "Mac Pitts: An Approach to Silicon Compilation", Computer, 1983.

[STA84] STAFF "Silicon Compilers Part 1: Drawing a Blank", VLSI Design, No.12, Dec 1984.

[SUN83] SUNGHO, K. "Linear Ordering and Application to Placement", Proceedings of the 20th Design Automation Conference, pp. 457-467, 1983.

[SUZ81] SUZIM, A. "Etude des Parties Operatives a Elements Modulares pour Processeur Monolitiques". Tese (Doct. Ing.) Institut National Polytechnique de Grenoble, Franca, 1981.

[SUZ86] SUZIM, A. "Sistema de CAD para Microeletrônica", Congresso da Sociedade Brasileira de Microeletrônica, pp. 341-350, Jul 1986.

[SZE82] SZEPIENIEC, A. "SAGA: An Experimental Silicon Compiler". Proceedings of the 19th Design Automation Conference, pp. 365-370, 1982.

[THO81] THOMAS, D. "The Automatic Synthesis of Digital System", Proceedings of the IEEE, Vol.69, No.10, Oct 1981.

[THO83] THOMAS, D. "Automatic Data Paths Synthesis", Computer, Dec 1983.

[THO82] THOMPSON, T. "A Utilitarian Approach to CAD". Proceedings of the 19th Design Automation Conference, pp. 23-29, 1982.

[TOD85] TODESCO, A. "Manual do Sistema RS", PGCC-UFRGS, Relatório Interno, 1985.

[TOD86] TODESCO, A. "Concepção de um Circuito Integrado do Tipo Processador com Conjunto de Instrucoes Reduzido". Diss Mest UFRGS-CPGCC, Porto Alegre, RS, 1986.

[TRA87] TRAYNOR, O. ; MALONE, J. "Design Rule Checking and VLSI", INTEGRATION, the VLSI journal, pp. 289-302, May 1987.

[TRE82] TRELEAVEN, P. "VLSI Processor Architecture", Computer 15, No.6, 1982.

[VAR85] VARINOT, P. ; CHUQUILLANQUI, S. "Method of PLA's Implementation: the Monoplane PLA", ESSIRC, 1985.

[WER82] WERNER, J. "The Silicon Compiler", VLSI Design, Sep 1982.

[WES85] WESTE, N. ; ESHRAGHIAN, K. "Principles of CMOS VLSI Design", ADDISON-WESLEY Publishing Company, 1985.

[WHI81] WHITNEY, T. "A Hierarquical Design-Rule Checking Algorithm", LAMBDA, First Quarter, 1981.

[WIL78] WILCOX, P. et al. "Design Rule Verification Based on One Dimensional Scans", Proceedings of the 15th Design Automation Conference, pp. 285-289, 1978.

[WIL83] WILLIANS, T. ; PARKER, K. "Design for Testability - A Survey", Proceedings of the IEEE, Vol.71, No.1, Jan 1983.

[WOO79] WOOD, R. "A High Density Programmable Logic Array Chip", IEEE Transactions on Computers, vol. C-28,No.9,pp. 602-608, 1979.

[YOS82] YOSHIMURA, T. "Efficient Algorithms for Channel Routing", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. CAD-1, No. 1, pp. 25-35, Jan. 1982.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CSAR: um compilador de silício voltado a execução de operações paralelas

Dissertação apresentada aos Srs.

Prof. Dr. Ricardo A. da L. Reis

Prof. Tiaraju Vasconcellos Wagner

Prof. Dr. Dante A. Couto Barone

Prof. Darlei Irineu Antunes de Abreu

Visto e permitida a impressão

Porto Alegre, 10/08/2006

Prof. Dr. Ricardo A. da L. Reis
Orientador

Prof. Dr. Ricardo Augusto da L. Reis
Coordenador do Curso de Pós-Graduação
em Ciência da Computação

Prof. Carlos Alberto Heuser
Coordenador do Programa de
Pós-Graduação em Computação-PPGC
Instituto de Informática - UFRGS