

Adriano Aparecido Tokunaga

Projeto de um Conjunto de Chaves
Programáveis para Uso em Barramentos de
Multiprocessadores



1910000757



" Projeto de um Conjunto de chaves
programáveis para uso em barramentos de
multiprocessadores "

**PROJETO DE UM CONJUNTO DE CHAVES
PROGRAMÁVEIS PARA USO EM BARRAMENTOS DE
MULTIPROCESSADORES**

Adriano Aparecido Tokunaga

Dissertação de Mestrado

**Pós-Graduação em Matemática Aplicada
e Computacional
MAC-018**

Dissertação apresentada à Unesp/Ibilce como parte
dos requisitos para a obtenção do título de Mestre
em Ciências Matemáticas - Área de Concentração
em Matemática Aplicada e Computacional

Data: 25 de fevereiro 1997



753/43

**“ Projeto de um Conjunto de chaves
programáveis para uso em barramentos de
multiprocessadores “**

Adriano Aparecido Tokunaga

Orientador: Prof. Dr. Norian Marranghello

Dissertação apresentada à Unesp/Ibilce como parte
dos requisitos para a obtenção do título de Mestre
em Ciências Matemáticas - Área de Concentração
em Matemática Aplicada e Computacional

Data: 25 de fevereiro 1.997



757/98

Agradecimentos

Agradeço ao Prof. Norian Marranghello pela orientação, pela confiança depositada, e conhecimento adquirido durante todo o decorrer deste trabalho de mestrado.

À FAPESP pelo apoio financeiro, concedido através do processo nº 95/01714-7

Agradeço a Marcelo e Fábio, da DSIF/FEE/UNICAMP, pelo auxílio na linguagem VHDL.

Aos colegas de mestrado, principalmente ao Wagner e Valéria pelo ambiente agradável de trabalho.

À minha namorada Deise, pela confiança, compreensão e apoio prestado em todo momento.

Aos meus pais, Mitsuru e Olga, meu irmão, Alexandre, pela incansável dedicação, compreensão e apoio na horas mais difíceis de trabalho.



Sumário

Uma forma de atender à necessidade de constantes aumentos na capacidade de processamento dos computadores é através do uso de multiprocessadores. Para viabilizar a reconfiguração da arquitetura dos multiprocessadores de modo a melhorar sua adequabilidade a certos problemas, bem como a aumentar sua tolerância a falhas, faz-se necessário o uso de chaves para a alteração dinâmica das rotas pelas quais os dados trafegam entre dois processadores quaisquer do arranjo.

Os objetivos deste trabalho são realizar o projeto lógico de um chave comutadora de barramentos e o projeto comportamental de um protocolo para a comunicação entre os processadores.

A chave projetada possui quatro canais de entrada e saída permitindo a conexão de processadores vizinhos, que estejam em um mesmo plano. O protocolo de comunicação visa sincronizar o tráfego dos dados pelo sistema, além de fazer a conversão dos dados paralelos gerados pelos processadores em seriais necessários para a transmissão pelo sistema de barramentos e vice-versa.

Ambas, chave e protocolo, foram descritos na linguagem VHDL e simulados. Como resultados destacam-se o tempo de atraso que as informações sofrem ao passarem pelas chaves e a especificação temporal dos sinais responsáveis pela sincronização das trocas de dados entre processadores.



Abstract

The meet the increasingly need for more computer processing power, multiprocessor can be used. To enable the reconfiguration of multiprocessor architectures so as to enhance their adequacy to certain kind of problem as well as to augment their fault tolerance, it is needed the use of switches to dynamically change the routes through which data flow between any two processors of the array.

The goals of this work are logically design a buss switching element and to develop the behavioral design of a protocol for the communication among processors.

The designed switch has four I/O channels, allowing the connection among neighboring processors, as long as they are in the same plane. The communication protocol is meant to synchronize the data flow through the system. Moreover, it has to convert parallel data generated by the processors into serial data needed for transmission through the buss system and vice-versa.

Both, switch and protocol, were described with the VHDL language, and then simulated. Results of these simulation promissing information delays through the switches as well as good synchronization signal timming for data exchange among processors.

ÍNDICE

Capítulo I - Introdução

1 - Origens do projeto	06
2 - Funcionamento	12

Capítulo II - Teoria Básica Relacionada

1 - Álgebra de Boole	16
2 - Realização Física dos Circuitos Integrados	23
3 - Projeto de Circuitos Integrados	37
4 - Projeto Lógico	41
5 - Simulação	52
6 - Síntese Automática	57
7 - Teste	60

Capítulo III - Projeto das Chaves

1 - Especificação do Projeto	61
2 - Projeto em Nível de Registradores	64
3 - Projeto em Nível de Portas Lógicas	67

Capítulo IV - Validação do Projeto

1 - A Descrição do Projeto em VHDL	71
2 - Análise Temporal do Sistema	91

Conclusão e Perspectivas

94

Referências Bibliográficas

95



Capítulo 1

Introdução

Este capítulo, está dividido em três itens: Origens do Projeto, Funcionamento e Aplicações e Estrutura da Dissertação.

No item 1, é apresentada uma descrição do sistema Simulador de Circuitos Baseado em *Hardware* (ABACUS), o qual é a fonte de inspiração do projeto das chaves. Apresenta-se, de forma concisa e bastante objetiva, o funcionamento geral da metodologia ABACUS. Dá-se destaque à forma como é realizada a configuração do arranjo de processadores, os quais são responsáveis pela emulação dos componentes de um circuito real durante o processo de simulação.

No item 2, descreve-se o funcionamento de uma chave. Descreve-se, também, o mecanismo de sincronização responsável pelas operações de leitura/escrita dentro dos processadores.

1 - Origens do Projeto

1.1 - Visão Geral da Arquitetura ABACUS [11]

A simulação de circuitos elétricos tem sido de muita importância para a engenharia elétrica, nos últimos anos. A densidade sempre crescente dos circuitos integrados tem tornado imperioso o uso de ferramentas de *software* no auxílio ao projeto e à fabricação de circuitos integrados. Entre estas ferramentas, podemos destacar os simuladores de circuito, que são ferramentas capazes de prever o comportamento elétrico de um circuito com detalhes suficientes para que se tenha uma idéia muito aproximada do seu comportamento real.

A metodologia ABACUS (*Hardware Based Circuit Simulator*) se aplica a construção de um simulador, ou seja, a fabricação de um equipamento dedicado à simulação de circuitos através de um arranjo especial de processadores. Este arranjo é composto por um processador mais complexo que serve de hospedeiro, ou gerente, do arranjo e por um grande número de processadores mais simples chamados de **MPH** (*Model Processing Hardware-element*), os quais incorporam modelos dos elementos de circuitos e sobre o qual é possível mapeá-los.

O processo inicial de simulação é realizado pelo processador hospedeiro, este lê a *netlist* de entrada e as demais informações referentes à simulação. A seguir, ele identifica os elementos de circuito, suas interconexões e as análises a serem feitas, para que tais elementos possam ser mapeados nos MPHs.



Este mapeamento é realizado graças a uma rede programável de barramentos, com **chaves** uniformemente distribuídas por todo o arranjo. Estas chaves deverão proporcionar o máximo de interconexões entre os MPHs, com um número razoável de barramentos e circuitos de chaveamento.

Após o mapeamento, o gerente transfere a cada MPH os parâmetros relativos ao elemento do circuito a ser simulado e às análises a serem feitas. Enquanto espera pelos resultados da simulação, o gerente, além de controlar a convergência global do arranjo, faz o tratamento dos dados disponíveis para a saída, ou seja, quando o arranjo chega a uma solução, o gerente para momentaneamente a simulação, recupera os resultados disponíveis e reinicia a simulação. Após todas as análises, o processador hospedeiro apresenta os dados ao usuário.

1.2 - Funcionamento dos MPHs [11]

O primeiro passo no processo de simulação de qualquer circuito eletrônico é, receber os dados de entrada fornecidos pelo usuário. Estes dados possibilitam ao simulador identificar os elementos do circuito, as análises a serem feitas, montar uma base de dados para o circuito simulado, montar um grafo de interconexão entre os MPHs, verificar o número de MPHs disponíveis, e particionar o arranjo, caso o número de componentes do circuito simulado seja maior que o número de MPHs disponíveis.

Após o gerente processar os dados de entrada fornecidos pelo usuário, o gerente irá transferir todas as informações necessárias ao arranjo de processadores (MPHs). Estas informações referem-se a:

1. interconexão entre os MPHs.
2. informações sobre qual o componente do circuito que cada MPH emulará.
3. estado inicial do arranjo.

O mapeamento dos elementos do circuito nos MPHs e as interconexões entre os MPHs só ocorre uma vez durante todo o processo de simulação.

Após serem mapeados e interconectados, os MPHs se encontram num estado desconhecido, ou seja necessitam ser inicializados.

Para ilustrar o funcionamento do arranjo, vamos definir **MPH0** como os MPHs que simulam as fontes de potência ou de sinal, ou seja, as entradas primárias. Os **MPH1** são os MPHs diretamente ligados às entradas primárias. Os **MPH2** são os MPHs ligados diretamente aos MPH1, por sua vez, os **MPH3** são os MPHs ligados diretamente aos MPH2, e assim sucessivamente.

Para iniciar o processo de simulação, os MPH0 colocam as entradas primárias em suas filas de saída para que possam ser acessados pelos MPH1. Após receberem estes dados, os MPH1 processam estes dados e realizam testes para validar estas informações e verificar suas convergências. Estes testes são baseados em valores pré-estabelecidos, quando da especificação das análises do circuito simulado. A partir destes testes, os MPH1 podem sinalizar ao gerente suas convergências. Depois do processamento e testes de validade, os MPH1 transferem estes resultados parciais para suas filas de



saída para que possam ser acessadas pelos MPH2 e MPH0, seus vizinhos, que por sua vez, irão realizar as mesmas operações que os MPH1.

Assim, os MPH0 e MPH2 irão realizar o processamento dos dados vindos dos MPH1, a seguir, irão realizar os testes de validade para verificar suas convergências, e finalmente, colocarão os resultados do processamento em suas filas de saída para que possam ser acessados pelos MPHs vizinhos.

É importante ressaltar que a validade dos testes de convergência nos MPHs serão sinalizadas ao gerente, sendo que o gerente só irá aceitar suas convergências se todos os outros MPHs do arranjo convergirem.

Desta forma, os dados válidos irão fluir por todo o arranjo de processadores, até que ao final do processamento, ou seja, quando os MPHs chegarem a valores estáveis, o ciclo será finalizado e os dados transferidos ao gerente.

Vários ciclos de processamento são necessários para realizar todas as análises do circuito simulado, sendo que cada ciclo de processamento é iniciado com novas entradas primárias.

O gerente só é responsável pela entrada e saída de dados do arranjo e também pelo controle da convergência global do sistema.

Em um circuito real, cada um de seus componentes trabalha independente dos demais, ou seja, o processamento interno de cada componente é realizado sem qualquer interferência dos demais componentes, eles apenas trocam informações. Isto também ocorre na arquitetura ABACUS, pois cada MPH irá trabalhar de forma assíncrona durante o processo de simulação, ou seja, o início e o final de um ciclo de processamento de um MPH é independente do processamento dos demais MPHs.

Resumindo, no ABACUS existem vários processadores onde se encontram os modelos que representam o comportamento dos diversos elementos do circuito. Estes processadores são interconectados através de um conjunto de barramentos programáveis. A seguir, os dados primários são fornecidos ao arranjo e a simulação é iniciada. A partir de um estado inicial, todos os processadores do arranjo começam a trabalhar simultaneamente, buscando dados em suas filas de entrada, processando-os e transferindo os resultados às filas de saída. Quando o arranjo se estabiliza para um dado conjunto de entrada, neste momento, as soluções são armazenadas, o tempo é incrementado e um novo conjunto de entradas primárias é aplicado ao circuito. O processo é repetido até que todos os instantes desejados sejam analisados.

É importante salientar que o fluxo de dados através do arranjo se dá de maneira semelhante aos sinais elétricos no circuito real. Neste esquema de processamento, os processadores trabalham totalmente de forma assíncrona, onde apenas os instantes inicial e final são controlados pelo hospedeiro.

1.3 - Especificação do ABACUS [11]

A arquitetura ABACUS deve possuir um computador hospedeiro composto de quatro processadores, cada um dos quais executando processos do hospedeiro e ligado diretamente aos outros três por meio de canais bidirecionais de alta velocidade, de modo a possibilitar o máximo de paralelismo. Estes canais devem ser constituídos por dois barramentos, cada



um transferindo informações num sentido. Esta elevada capacidade de transferência de informação, deve-se à necessidade de um fluxo rápido das informações entre os processadores.

A memória de cada um destes processadores deve ser suficientemente grande para permitir o armazenamento de todas as informações referentes ao circuito proposto para simulação.

O processador responsável pelo controle do computador hospedeiro terá uma carga computacional menor, portanto, deverá também fazer a interface com o usuário.

O processador responsável pelo gerenciamento do arranjo terá canais especiais para a transferência de dados entre o hospedeiro e o arranjo, controle do arranjo e programação das **chaves dos barramentos do arranjo**.

Um arranjo de processadores para processar os modelos dos componentes de um circuito físico. O ideal seria que o número de processadores fossem no mínimo igual ao número de elementos do circuito a ser analisado. Cada processador do arranjo deve ter uma memória RAM, para o armazenamento dos resultados parciais da simulação, uma memória ROM para o armazenamento de microprogramas correspondentes aos elementos do circuito a ser simulado e *buffers* de entrada e saída para a comunicação entre os processadores do arranjo e o hospedeiro.

Um exemplo seria utilizar um arranjo de 32.768 (2^{15}) processadores, dispostos em um circuito com 32 camadas, onde cada camada contenha 32 x 32 processadores.

Através da figura 1, pode-se ver claramente a disposição das chaves em relação aos barramentos e aos MPHs. Existem três tipos de chaves, as quadradas, as pentagonais e as hexagonais cada uma desempenhando uma função diferente na troca de dados entre os MPHs do arranjo. Na realidade, as chaves é que irão estabelecer um caminho de comunicação através dos barramentos para que os dados sejam acessados somente pelos MPHs requeridos.

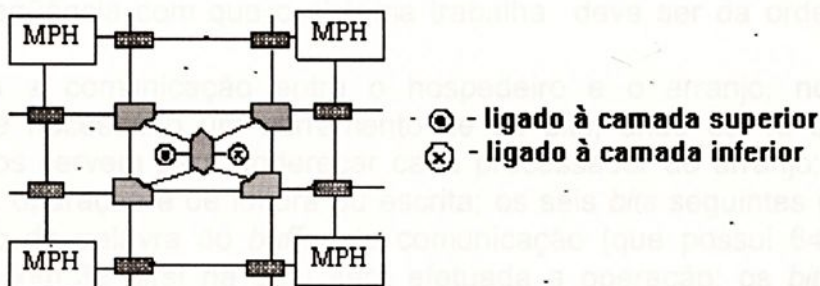


Figura 1 - Disposição dos MPHs, barramentos e chaves

As ligações entre os barramentos podem ser realizadas tanto no plano horizontal como no vertical, por meio de blocos de chaves localizadas nas regiões de intersecção dos barramentos, ou seja, no cruzamento das linhas imaginárias que unem os vértices dos processadores de um mesmo plano horizontal. A figura 2 apresenta uma visão tridimensional de uma chave hexagonal utilizada na interligação das diversas camadas de processadores.

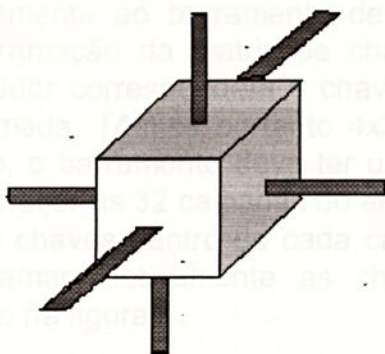


Figura 2 - Esquema tridimensional de uma chave

As memórias RAM, dos processadores do exemplo, deverão ter 1 KB, sendo:

- 256 *bytes* para o *buffer* de comunicação com o hospedeiro;
- 256 *bytes* referente às informações do componente e da análise;
- 128 *bytes* vindas dos demais processadores e para os resultados parciais da simulação;
- 384 *bytes* para rascunho durante a execução;

As memórias ROM devem possibilitar o armazenamento de microprogramas, com modelos de elementos de um circuito. Seu tamanho deve ser tal que possibilite armazenar os elementos de um circuito. Estas memórias devem possibilitar a alteração dos modelos gravados, a inclusão de novos modelos e a remoção de modelos em desuso.

O número de canais de um processador do arranjo é definido em função da conectividade dos nós do circuito típico. A simetria quadrangular do arranjo é devida à utilização de quatro filas, ou canais, de entrada e saída (FES) no MPH. Este número de filas é adotado porque dificilmente convergirão mais do que quatro elementos a um MPH. Há de se notar que sobre um MPH ainda converge um quinto canal implementado através de um *buffer* de 256 *bytes*, este realizando a troca de dados entre o MPH e o computador hospedeiro.

A frequência com que o sistema trabalha deve ser da ordem de 100 MHz.

Para a comunicação entre o hospedeiro e o arranjo, no exemplo proposto, é necessário um barramento de 56 *bits*, onde os 15 *bits* menos significativos servem para endereçar cada processador do arranjo; o 16º *bit* indica se a operação é de leitura ou escrita; os seis *bits* seguintes constituem o endereço da palavra do *buffer* de comunicação (que possui 64 palavras, cada uma com 32 *bits*) na qual será efetuada a operação; os *bits* 22 a 53 contêm a informação a ser transferida entre o processador do arranjo e o hospedeiro, e os dois restantes informam a condição de convergência do arranjo e o controle de sua operação. O esquema desta palavra é mostrado na figura 3.

0	14	15	16	21	22	53	54	55
Endereço do Processador	L	Palavra	Dados a serem transferidos	C	P			
	/	do	de ou	O	A			
	E	"Buffer"	para o computador	N	R			

Figura 3 - Esquema da palavra de comunicação de dados



Operando paralelamente ao barramento de dados deve-se ter outro barramento para a programação da matriz de chaves dos barramentos do arranjo. A cada processador correspondem 4 chaves nas linhas e 4 chaves nas colunas de cada camada. Têm-se portanto $4 \times 32 = 128 = 2^7$ chaves por linha ou coluna. Portanto, o barramento deve ter uma largura de 34 *bits* dos quais 5 servem para endereçar as 32 camadas do arranjo, 7 para endereçar as linhas e 7 as colunas de chaves dentro de cada camada. Os demais 15 *bits* são usados para programar efetivamente as chaves. O esquema deste barramento é apresentado na figura 4.

0	4 5	11 12	18 19	33
Endereço da Camada	Endereço da Linha	Endereço da Coluna	Programação das Chaves	

Figura 4 - Esquema da palavra de programação das chaves

A interconexão entre chaves e processadores (MPH) pode ser visto no esquema da figura 5.

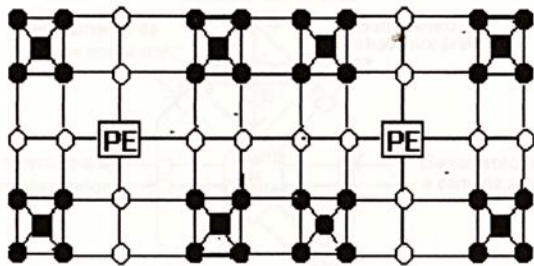


Figura 5 - Esquema plano da disposição tridimensional das chaves

A partir da figura 5, podemos ver que existem três tipos de chaves para a interconexões dos MPHs, assim temos:

1. Círculos vazados - correspondem às chaves quadradas intra-camadas, que servem para interligar os processadores do arranjo aos barramentos adjacentes e/ou outros processadores vizinhos a eles, dentro de uma mesma camada, figura 6.
2. Círculos cheios - correspondem às chaves pentagonais intra-camadas, que servem para a interligação dos processadores com os vizinhos mais distantes, para as interligações norte-sul (NS) com as leste-oeste (LO) ou para a conexão das rotas planares (NS/LO) com as chaves de roteamento inter-camadas, figura 7.



3. Quadrados cheios - chaves hexagonais inter-camadas, permitem a ligação entre os processadores nas diversas camadas do arranjo, figura 7.

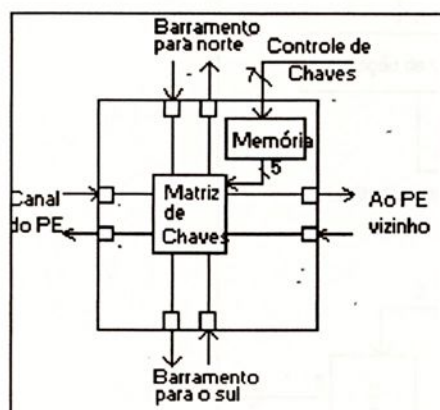


Figura 5 - Chave Quadrada

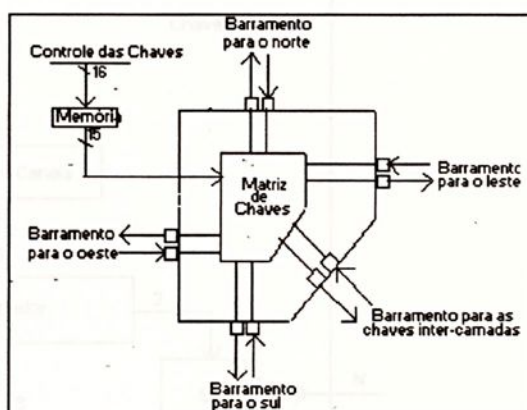


Figura 6 - Chave Pentagonal

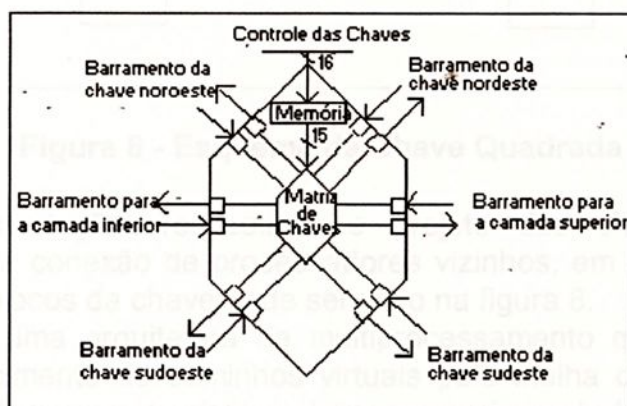


Figura 7 - Chave Hexagonal

Todas estas chaves possuem uma pequena memória, o que permite que sejam programadas no início do processamento, para que possam interligar os MPHs para a simulação do circuito proposto.

2 - Funcionamento

2.1 - As Chaves

A função das chaves em uma arquitetura de multiprocessamento é a de possibilitar a configuração e reconfiguração dos caminhos por onde a informação circula, ou seja, a informação vinda de um processador, A, pode ser encaminhada a um processador B em um momento e, em outro a C, por

meio das chaves. Futuramente, a mesma informação vinda de A, pode ser repassada a um processador D, posteriormente a C, por meio de reconfiguração das chaves.

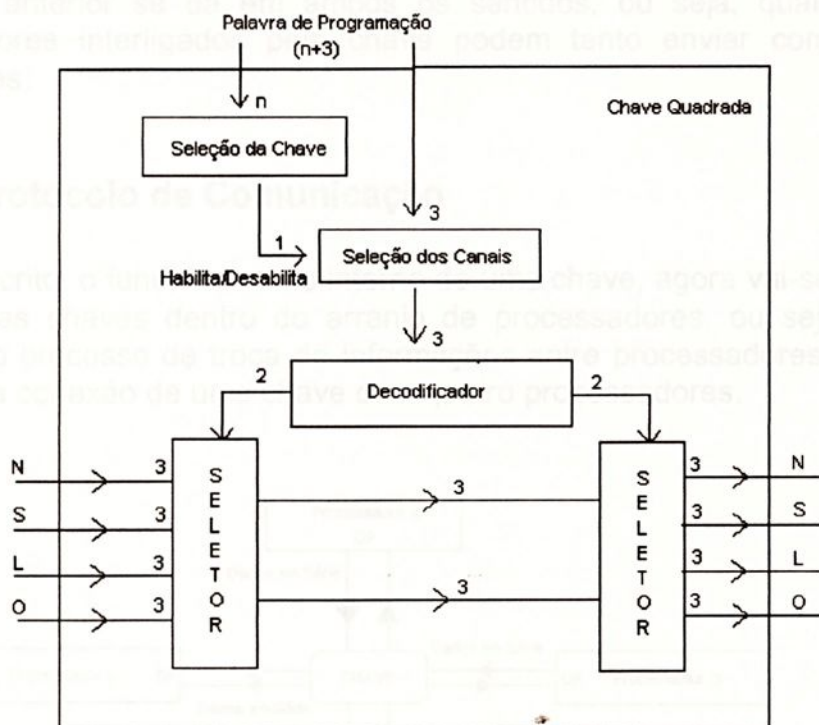


Figura 8 - Esquema da Chave Quadrada

Nesta dissertação, realizou-se o projeto das chaves quadradas, responsáveis pela conexão de processadores vizinhos, em um mesmo plano. O diagrama em blocos da chave pode ser visto na figura 8.

Para que uma arquitetura de multiprocessamento que utiliza chaves para o estabelecimento de caminhos virtuais pela malha de processadores, possa trabalhar, faz-se necessário um barramento de controle por onde trafega o controle das chaves.

Este controle é responsável pela seleção das chaves e, posteriormente, pela habilitação dos canais por onde trafegam os dados vindos dos processadores. Vale lembrar que em uma chave convergem quatro canais de entrada e saída, assim, a função da chave é possibilitar que dois dos quatro canais troquem informações.

Como especificado na metodologia ABACUS, a palavra de controle, tem dois tipos de informação, a especificação do endereço da chave e o controle de habilitação dos canais. Supõem-se uma palavra com $(n+3)$ bits, os n primeiros destinam-se à seleção da chave dentro do arranjo de chaves e os 3 últimos bits à conexão dos canais dentro da chave.

A especificação do número correto de bits da palavra de controle será apresentada no capítulo 4.

Quando a palavra de controle chega na chave, faz-se inicialmente o teste de endereçamento. Se o endereço contido na palavra de controle é o mesmo que o da chave, os bits de controle de habilitação dos canais de comunicação serão repassados à chave. Estes bits de controle, são

transformados em um novo conjunto de informações, o qual será usado para selecionar dois dos quatro canais ligados a chave a se comunicarem.

É importante destacar que a habilitação dos canais mencionados no parágrafo anterior se dá em ambos os sentidos, ou seja, quaisquer dois processadores interligados pela chave podem tanto enviar como receber informações.

2.2 - O protocolo de Comunicação

Descrito, o funcionamento interno de uma chave, agora vai-se estudar a atuação das chaves dentro do arranjo de processadores, ou seja, como é realizado o processo de troca de informações entre processadores. Na figura 9, tem-se a conexão de uma chave com quatro processadores.

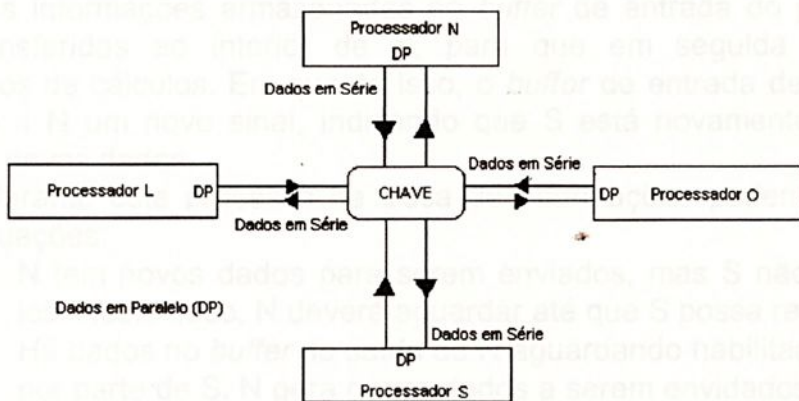


Figura 9 - Integração dos Processadores com a chave

Em sistemas de multiprocessamento abordados nesta dissertação, a tarefa inicial é a programação da matriz de chaves, ou seja, é o estabelecimento de caminhos por onde a informação trafegará.

Estes caminhos são estabelecidos após a chegada de palavras de controle nas chaves do arranjo. Estas palavras irão configurar algumas das chaves do arranjo, de maneira que se estabeleça uma rota por onde a informação passará.

Realizada a configuração das chaves, os processadores do arranjo começam a trabalhar e enviar seus cálculos aos processadores vizinhos, por meio da chave.

Vamos abstrair do arranjo de processadores e chaves, um pequeno conjunto composto por 1 chave e 2 processadores, N e S.

As tarefas desempenhadas por N e S são: a realização do processamento interno das informações vindas de seus *buffers* de entrada; e a colocação dos dados processados em seus *buffers* de saída. Os dados nos *buffers* de saída são, oportunamente, enviados aos processadores vizinhos.



Considerando-se que a chave já esteja configurada para realizar a comunicação de N e S, passa-se ao estágio de processamento e troca de dados.

Vamos iniciar o processo com N, ou seja, N realiza seus cálculos internos, e em seguida coloca as informações processadas em seus *buffers* de saída. Estes dados armazenados nos *buffers* de N estão agora disponíveis a S.

O processador N só irá enviar o conteúdo de seu *buffer* de saída quando receber, de S, um sinal indicando que o mesmo pode receber novos dados.

Assim que receber o sinal, N irá converter seu dados, normalmente em paralelo, para seriais, afim de que eles possam ser transmitidos.

Quando chegam em S, os dados são convertidos em paralelos e colocados nos *buffers* de entrada de S, para que possam ser processados. Juntamente com a conversão, é enviado ao processador N, um sinal indicando que S está atualmente impossibilitado de receber novos dados.

As informações armazenadas no *buffer* de entrada do processador S, são transferidas ao interior de S, para que em seguida sofram novos processos de cálculos. Em quanto isso, o *buffer* de entrada de S é limpo e é enviado a N um novo sinal, indicando que S está novamente capacitado a receber novos dados.

Durante este processo de troca de informações podem ainda ocorrer duas situações:

- N tem novos dados para serem enviados, mas S não pode recebê-los. Neste caso, N deverá aguardar até que S possa recebê-los.
- Há dados no *buffer* de saída de N aguardando habilitação para envio, por parte de S. N gera novos dados a serem enviados para S. Neste caso, N escreve os novos dados sobre aqueles existentes no *buffer* de saída, desprezando a informação anterior.

1 - Álgebra de Boole [1,2]

1.1 - Introdução

A Álgebra de Boole é a responsável pela base matemática de um projeto lógico, proporcionando uma especificação exata e clara da operação realizada.

A Álgebra de Boole surgiu de maneira curiosa, a partir de uma controvérsia entre dois matemáticos De Morgan e Hamilton. Boole, amigo de De Morgan, para defendê-lo escreveu um artigo intitulado *The Mathematical Analysis of Logic*, o qual originou a Álgebra de Boole conhecida atualmente.



Capítulo 2

Teoria Básica Relacionada

Neste capítulo, é apresentada toda a teoria básica estudada durante o decorrer do mestrado.

No item 1 é descrita a Álgebra de Boole, a qual é uma ferramenta importante dentro do projeto de circuitos lógicos. Proporciona ao projetista a base matemática necessária à descrição clara e objetiva das funções desempenhadas pelo circuito a ser projetado.

No item 2 descreve-se o processo para obtenção das lâminas de silício. Este item mostra o porque da escolha deste elemento químico, como ocorre a sua purificação e sua posterior transformação em lâminas. Também são apresentados os principais processos que transformam lâminas de silício em circuitos integrados. Adicionalmente, é mostrado em nível atômico, o princípio de funcionamento dos circuitos integrados e, finalmente, quais as principais tecnologias de fabricação.

No item 3, é dada uma visão geral de como é realizado o projeto de um circuito integrado.

No item 4, é dado um enfoque especial ao projeto lógico. Expõem-se, de forma objetiva, os conceitos deste nível de projeto e os principais tipos de ferramentas empregadas na descrição de um circuito. Ademais, são descritos alguns dos circuitos lógicos mais utilizados, tais como *flip-flops*, *multiplexadores*, *registradores*, etc.

No item 5, descreve-se o processo de simulação de um circuito eletrônico. São apresentados os principais tipos de simuladores, tais como os simuladores funcional e o lógico.

No item 6, apresenta-se o processo de síntese funcional.

No item 7, enfoca-se a etapa de testes que deve ser realizada em um circuito eletrônico integrado, após a sua implementação física.

1 - Álgebra de Boole [13]

1.1 - Introdução

A Álgebra de Boole é a responsável pela base matemática de um projeto lógico, proporcionando uma especificação exata e clara da operação realizada.

A Álgebra de Boole surgiu de maneira curiosa, a partir de uma controvérsia entre dois matemáticos De Morgan e Hamilton. Boole, amigo de De Morgan, para defendê-lo escreveu um artigo intitulado *The Mathematical Analysis of Logic*, o qual originou a Álgebra de Boole conhecida atualmente.



A Álgebra de Boole pode ser restrita a um conjunto de elementos (0 e 1), sobre estes elementos são aplicadas operações elementares, como a adição, a multiplicação e a negação. Cada uma destas operações, juntamente às variáveis, as quais assumem os valores binários, dão origem às chamadas equações booleanas.

Uma equação booleana descreve, matematicamente, o comportamento de um circuito eletrônico, ou seja, é possível prever seu comportamento funcional em termos da equação booleana correspondente.

Dependendo da equação booleana que descreve matematicamente o circuito, o projetista tem condições de simplificá-la, ou seja, aplicar os vários postulados e teoremas da Álgebra de Boole e transformar a equação original em uma nova equação, com as mesmas características da original, mas, com uma representação mais simples de se converter em um circuito físico.

1.2 - Conjuntos Parcialmente Ordenados

Um conjunto parcialmente ordenado é uma estrutura matemática, na forma, $(X, \geq)$ onde X é um conjunto não vazio (finito ou infinito) e $\geq$ é uma relação (que pode ser dita “maior que ou igual à”, “contêm”, ou simplesmente “ $\geq$ ”) que verifica as seguintes propriedades:

a) Reflexiva:

$$x \geq x, \forall x \in X.$$

b) Anti - Simétrica

$$x \geq y \text{ e } y \geq x \rightarrow x = y, \forall x, y \in X$$

c) Transitiva

$$x \geq y \text{ e } y \geq z \rightarrow x \geq z, \forall x, y, z \in X.$$

Exemplo: Considere $X = \mathbb{Z}$ (conjunto dos números reais), e a relação $\leq$, então $(\mathbb{Z}, \leq)$ é um conjunto parcialmente ordenado, pois:

a) $n \leq n, \forall n \in \mathbb{Z}$;

b) $n \leq m \text{ e } m \leq n \rightarrow n = m, \forall m, n \in \mathbb{Z}$

c) $n \leq m \text{ e } m \leq p \rightarrow n \leq p, \forall m, n, p \in \mathbb{Z}$

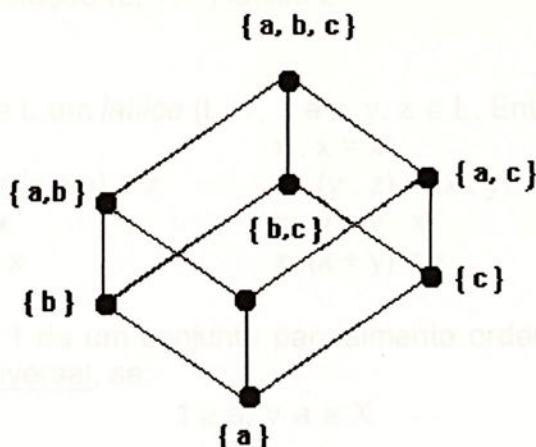
1.3 - Diagrama de Hass

Diagrama de Hass é uma representação gráfica de um conjunto parcialmente ordenado, X , com um número finito de elementos.

Exemplo: Considere o conjunto parcialmente ordenado:

$P(S) = \{\text{todos os subconjuntos de } S\}$ onde $S = \{a, b, c\}$.

Seja a relação $A \geq B \leftrightarrow A \supseteq B, \forall A, B \in P(S)$. Então o diagrama de Hass, para este exemplo, é mostrado no esquema 1.



Esquema 1 - Diagrama de Hass

1.4 - Lattice

Seja um conjunto parcialmente ordenado $(X, \geq)$ e os elementos $x, y \in X$.

Um elemento $u \in X$ é chamado menor limitante superior (MLS) de x e y se:

a) $u \geq x$, e $u \geq y$; (1)

b) $\exists w \in X$, tal que $w \geq x$, $w \geq y$ e $w \geq u$ (2)

Um elemento $v \in X$ é chamado maior limitante inferior (MLI) de x e y se:

a) $v \leq x$, $v \leq y$; (3)

b) $\exists w \in X$, tal que $w \leq x$, $w \leq y$ e $w \leq v$ (4)

Exemplo: Seja o conjunto parcialmente ordenado $(\mathbb{Z}^+, |)$, onde $|$ é uma relação de divisibilidade. Tomemos os elementos $2, 3 \in \mathbb{Z}^+$. Então:

- $\text{MLS}(2,3) = \text{mmc}\{2,3\} = 6$, onde o mmc é o mínimo múltiplo comum.

- $\text{MLI}(2,3) = \text{mdc}\{2,3\} = 1$, onde o mdc é o máximo divisor comum.

Teorema: Seja $(X, \geq)$ um conjunto parcialmente ordenado, e sejam $x, y \in X$. Então, se x e y tem MLI e MLS, respectivamente, u e v . Então u e v são únicos.

Um lattice é um conjunto parcialmente ordenado $(L, \geq)$ em que cada par de elementos $x, y \in L$ tem:

I) um MLS, que é denotado por $x + y$ ou $x \vee y$; e

II) um MLI, que é denotado por $x \cdot y$ ou $x \wedge y$;

Assim, podemos reescrever as definições em (1), (2), (3) e (4), acima, respectivamente como segue:

1) $x + y \geq x$, $x + y \geq y$

2) $z \geq x$, $z \geq y \rightarrow z \geq x + y$

3) $x \cdot y \leq x$, $x \cdot y \leq y$

4) $z \leq x$, $z \leq y \rightarrow z \leq x \cdot y$

Usaremos a notação $(L, +, \cdot)$ *lattice* L

Teorema: Seja L um *lattice* $(L, +, \cdot)$ e $x, y, z \in L$. Então:

a) $x + x = x$

$x \cdot x = x$

b) $x + (y + z) = (x + y) + z$

$x \cdot (y \cdot z) = (x \cdot y) \cdot z$

c) $x + y = y + x$

$x \cdot y = y \cdot x$

d) $x + (x \cdot y) = x$

$x \cdot (x + y) = x$

Um elemento 1 de um conjunto parcialmente ordenado $(X, \geq)$, será um limitante superior universal, se:

$$1 \geq a, \forall a \in X$$

Um elemento 0 de um conjunto parcialmente ordenado $(X, \geq)$ será um limitante inferior universal, se:

$$0 \leq a, \forall a \in X$$

Então 0 é chamado identidade para "+", e 1 é chamado identidade para " $\cdot$ ". Logo:

a) $a + 1 = 1$

$a \cdot 0 = 0$

b) $a \cdot 1 = a$

$a + 0 = a$

Definição: Um *lattice* será distributivo se as seguintes propriedades valerem:

a) $x + (y \cdot z) = (x + y) \cdot (x + z)$

b) $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

Definição: Um *lattice* $(L, +, \cdot)$ será complementado se para cada $a \in L$, existir um elemento a^* tal que:

a) $a + a^* = 1$;

b) $a \cdot a^* = 0$.

1.5 - Formalização Matemática da Álgebra de Boole

Uma Álgebra de Boole é um *lattice* $(L, +, \cdot)$ com:

a) 1 e 0 elementos universais, respectivamente, superior e inferior;

b) verifica-se a propriedade distributiva; e

c) verifica-se a propriedade de complementação ($-$);

Podemos denotar uma Álgebra de Boole por $(L, +, \cdot, -, 0, 1)$.

Propriedades: Seja $(L, +, \cdot, -, 0, 1)$ uma álgebra de Boole, então:

a) $(a+b)^* = a^* \cdot b^*$

b) $(a \cdot b)^* = a^* + b^*$

c) $a^{**} = a$

Definição: Uma álgebra de Boole de dois elementos é um a álgebra onde o conjunto L é:

$$L = \{ 1, 0 \}$$

Definição: Um polinômio booleano é uma expressão $f(x_1, x_2, \dots, x_n)$, onde $x_i, i = 1, \dots, n$ pertence a uma álgebra de Boole, em especial a uma de dois elementos. Portanto, $f(x_1, x_2, \dots, x_n)$ pode ser:

- I) um elemento x_i ; ou
- II) o produto de dois polinômios booleanos; ou
- III) a soma de dois polinômios booleanos; ou
- IV) o complemento de um polinômio booleano;

Definição: Cada elemento, x_i , de uma álgebra de Boole tem dois literais, x_i e x_i^* .

Teorema de Morgan: Para se complementar um polinômio booleano basta trocar (+) por (.), e vice-versa, e também trocar x_i por x_i^* .

Propriedades:

- a) $f(x_1, x_2, \dots, x_n)^* = (x_1 + x_2 + \dots + x_n)^* = x_1^* \cdot x_2^* \cdot \dots \cdot x_n^*$
- b) $f(x_1, x_2, \dots, x_n)^* = (x_1 \cdot x_2 \cdot \dots \cdot x_n)^* = x_1^* + x_2^* + \dots + x_n^*$

Teorema: $\forall f(x_1, x_2, \dots, x_n)$ vale:

- a) $f(x_1, x_2, \dots, x_i, \dots, x_n) = f(x_1, x_2, \dots, 1, \dots, x_n) \cdot x_i + f(x_1, x_2, \dots, 0, \dots, x_n) \cdot x_i^*$
- b) $f(x_1, x_2, \dots, x_i, \dots, x_n) = [f(x_1, x_2, \dots, 0, \dots, x_n) + x_i] \cdot [f(x_1, x_2, \dots, 1, \dots, x_n) + x_i^*]$

Definição: Um mintermo ou produto fundamental é uma conjunção de literais, ou seja, o produto de literais em que cada literal aparece exatamente uma vez.

Exemplo: $x_1 \cdot x_2 \cdot x_3$ é mintermo, mas $x_1 \cdot x_2 \cdot x_1^* \cdot x_3$ não é mintermo.

Definição: Um maxtermo ou soma fundamental é uma disjunção de literais, ou seja, a soma de literais em que cada literal aparece exatamente uma vez.

Exemplo: $x_1 + x_2 + x_3$ é maxtermo, mas $x_1 + x_2 + x_1^* + x_3$ não é maxtermo.

1.6 - Formas Disjuntivas e Conjuntivas Normais

Uma forma disjuntiva normal de uma expressão booleana tem a forma:

$$f(x_1, x_2, \dots, x_n) = C_1 + C_2 + C_3 \dots + C_n$$

onde $c_i, i = 1, \dots, n$ são mintermos.



Uma forma conjuntiva normal de uma expressão booleana é da forma:

$$f(x_1, x_2, \dots, x_n) = C_1 \cdot C_2 \cdot C_3 \cdot \dots \cdot C_n$$

onde c_i , $i = 1, \dots, n$ são *maxtermos*.

Definição: O dual de uma função booleana $f(x_1, x_2, \dots, x_n)$ é definido como $f^*(x_1^*, x_2^*, \dots, x_n^*)$ onde f^* é a inversão, complemento ou negação da função f .

Assim, $f^* = [f(x_1, x_2, \dots, x_n)]^*$.

Exemplo: Seja a função $f(x_1, x_2, x_3, x_4) = x_1 x_2 \vee x_1 x_3 \vee x_2 x_3 x_4$. Então f^* será:

$$f^* = (x_1 \vee x_2) (x_1 x_3) (x_2 \vee x_3 \vee x_4)$$

1.7 - Relação de Implicação e Primo Implicante

Definição: Dadas duas funções (expressões) booleanas $f(x)$ e $g(x)$, onde $x \in L$, L sendo uma álgebra Boole, diz-se que **f implica g** , e denota-se pôr $f \subseteq g$, se:

$$\forall x \in L, f(x) = 1 \rightarrow g(x) = 1.$$

Definição: **f implica estritamente g** se $\exists x \in L$, tal que $f(x) = 0 \rightarrow g(x) = 1$.

Definição: Duas funções f e g são comparáveis se $f \subseteq g$ ou $g \subseteq f$, caso contrário são incomparáveis.

Definição: Um implicante de uma função booleana é um *termo* que implica f , por *termo* entende-se uma conjunção ou disjunção de literais.

Exemplo: $f(x_1, x_2, x_3) = x_1 x_2 \vee x_2 x_3$.

Os termos são: x_1 , x_2 , x_3 , $x_1 x_2$, $x_1 x_3$, $x_2 x_3$.

Definição: Um *termo* P inclui outro *termo* Q se o conjunto de literais que compõem o termo Q for um subconjunto dos componentes de P .

Exemplo: O termo $P = x_1 x_2^* x_3 x_4^*$ inclui o termo $Q = x_1 x_2^* x_4^*$

Definição: Um **primo implicante** de uma função booleana, f , é um *termo* que implica f , tal que, não inclui outro implicante em f .

Exemplo: $f(x_1, x_2) = x_1 x_2 \vee x_2 x_3$

O primo implicante de f é $x_1 x_2$.

1.8 - Método do Consenso

Definição: O consenso dos termos P e Q se define como a conjunção dos termos $P' Q'$, onde P' e Q' são obtidos eliminando-se um elemento, x_i em P , e seu complemento, x_i^* , de Q .

Em outras palavras os termos P' e Q' são expressões exatamente na forma:

$$P = xP' \quad \text{e} \quad Q = x^*Q'$$

A conjunção $P'Q'$ é chamada **consenso de PQ**.

Exemplo: O consenso dos termos $P = x_1x_2^*x_3$ e $Q = x_1^*x_2^*x_4x_5$ é:

$$P'Q' = x_2^*x_3x_4x_5$$

Teorema: Se 2 termos P e Q possuem consenso, então seu consenso implica a **disjunção dos termos** P e Q . Assim, o método consiste em:

1. eliminar um termo que contiver outro termo na função dada; e
2. se P e Q possuem consenso, adicionar o consenso deles à função dada.

O resultado deste método é uma simplificação da função (expressão) booleana. Ele produz resultados semelhantes aos produzidos pelos mapas de Karnaugh.

Uma grande vantagem em relação a Karnaugh é que o método do consenso não se torna mais confuso a medida que o número de variáveis aumenta, como ocorre com o método de Karnaugh.

Exemplo: $f = x_1^*x_2^* + x_1x_2x_3 + x_1^*x_2x_3 + x_1x_3^*$

- a) escolhemos os termos $x_1^*x_2^*$ e $x_1x_3^*$ de f , e calculamos o consenso entre eles. Este consenso é $x_2^*x_3^*$;
- b) adicionamos a f o consenso calculado em a) e obtemos uma nova função $f = x_1^*x_2^* + x_1x_2x_3 + x_1^*x_2x_3 + x_1x_3^* + x_2^*x_3^*$;
- c) calculamos agora o consenso entre $x_1x_2x_3$ e $x_1^*x_2x_3$. Este consenso é x_2x_3 .
- d) adicionamos a f o consenso calculado em b) e obtemos uma nova função $f = x_1^*x_2^* + x_1x_2x_3 + x_1^*x_2x_3 + x_1x_3^* + x_2^*x_3^* + x_2x_3$;
- e) eliminamos o termo $x_1x_2x_3$, pois ele contém o termo x_2x_3 ;
- f) eliminamos o termo $x_1^*x_2x_3$, pois ele contém o termo x_2x_3 ;
- g) após eliminarmos estes termos, temos $f = x_1^*x_2^* + x_1x_3^* + x_2^*x_3^* + x_2x_3$;
- h) calculamos o consenso entre os termos $x_1x_3^*$ e x_2x_3 . Este consenso é x_1x_2 .
- i) adicionamos à função f o consenso calculado em h) e obtemos $f = x_1^*x_2^* + x_1x_3^* + x_2^*x_3^* + x_2x_3 + x_1x_2$

Após a aplicação do método do consenso, pode-se observar que a função f se tornou mais simples, pois os termos constituídos de três literais passaram agora a apenas 2 literais.



2 - Realização Física dos Circuitos Integrados

2.1 - Fabricação de Lâminas de Silício [12]

2.1.1 - Introdução

Os elementos da tabela periódica podem ser divididos em metais, não-metais e semi-metais. Os semi-metais têm características físico-químicas que estão entre as dos metais e as dos não-metais. Analogamente aos semi-metais, os semi-condutores têm condutividade intermediária à dos condutores e à dos isolantes. Tal condutividade é definida em função da quantidade de elétrons livres disponíveis para compor a corrente. Os semicondutores mais comumente usados na construção de circuitos integrados (CIs) são o germânio e o silício.

A industrialização dos semicondutores se deu na década de 1950, sendo o germânio (Ge) o primeiro material utilizado para a fabricação de circuitos integrados. As razões que levaram à utilização do germânio foram:

- seu ponto de fusão relativamente baixo;
- sua fácil purificação;
- possibilidade de seu crescimento em cadinhos de grafite;

Já o silício (Si), ao contrário do germânio, possui um ponto de fusão quase quinhentos graus acima do Ge e no estado líquido é muito reativo, exigindo técnicas mais refinadas para suas purificação e cristalização, técnicas estas não disponíveis a 40 anos atrás.

Todavia, o Si não foi totalmente descartado, pois o Ge tem algumas propriedades inconvenientes, tais como sua estabilidade com a temperatura inferior à do Si. Além disso, os óxidos de Ge são muito voláteis, o que não ocorre com os de Si.

O Si é de fácil oxidação, produzindo filmes finos de óxidos com ótimas propriedades mecânicas, químicas e isolantes. Os óxidos de silício têm aplicações importantes, particularmente na microeletrônica, onde as principais são o mascaramento de áreas de difusão, o isolamento dos dispositivos dos circuitos integrados, a proteção das regiões ativas dos CIs e como material ativo nos dispositivos fabricados com a tecnologia **MOS**.

Além do Si e do Ge, outros elementos químicos foram utilizados como semi-condutores, mas logo foram descartados. A principal razão foi a escassez de matéria prima.

Ademais, depois do oxigênio, o silício é o principal elemento constituinte da crosta terrestre, responsável por 27,7% do peso da litosfera. O Si é encontrado na natureza em duas formas, quais sejam: as sílicas e os silicatos.

As sílicas (SiO_2) compõem diversos minerais, tais como o quartzo. Já os silicatos são composições de silício principalmente com metais, oxigênio, hidrogênio e flúor.

Os silicatos são mais abundantes que as sílicas, compondo, por exemplo, o feldspato, a mica, a turmalina, o talco, o topázio, as argilas e as terras ácidas.



Por toda esta abundância e por suas características convenientes, o silício tornou-se o principal elemento na fabricação de circuitos integrados a semicondutores. Porém, para que a indústria de microeletrônica possa utilizá-lo com esta finalidade, o Si deve passar por um número de processos de purificação. Estes processos é o que veremos a seguir.

2.1.3 - Obtenção do Silício de Grau Eletrônico

2.1.2 - Obtenção do Silício de Grau Metalúrgico

O silício natural, como acabamos de ver, é encontrado em combinação com diversos outros elementos químicos. Precisamos então separá-lo destes elementos para podermos utilizá-lo na indústria.

Uma maneira de se obter o silício é a partir das sílicas. Normalmente, isto é feito colocando-se num forno de alvenaria, revestido com tijolos de grafite, quartzo e/ou quartzita, carvão mineral, coque e lenha, tudo triturado. A mistura é aquecida a cerca de 1700°C por meio de eletrodos de grafite. Este forno é mostrado na figura 10.

A reação produz, junto à parede inferior do forno uma crosta de *carborundum* (SiC), um material muito duro, de difícil limpeza, tendo um ponto de fusão superior a 2200°C. Uma camada intermediária com retenção de SiO₂, C e um pouco de SiC forma-se junto aos eletrodos de grafite. Vários gases são eliminados, tais como CO, CO₂, SiO e H₂O. Todo este silício, eliminado em forma de óxidos e carbetos, representam perdas no processo de obtenção do silício de grau metalúrgico (**Si - GM**). O rendimento final deste processo, se bem controlado, pode chegar a 70% e o silício obtido tem uma pureza de cerca de 98%, podendo chegar a 99,5% se a matéria prima utilizada for escolhida adequadamente.

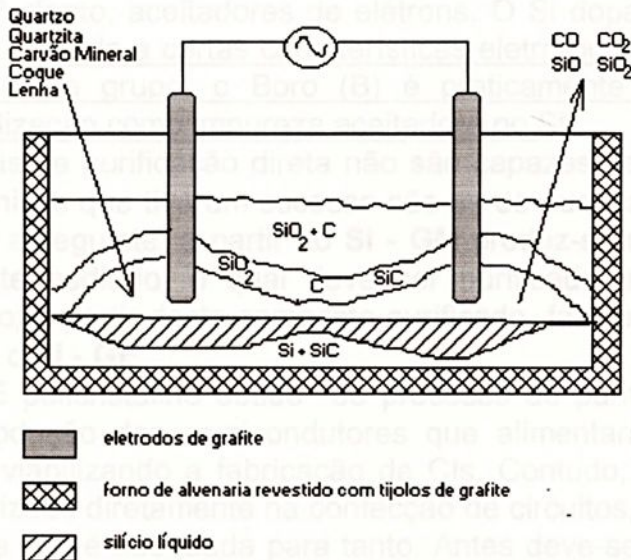


Figura 10 - Esquema para a Obtenção de Si - GM

Entretanto, este silício não é suficientemente puro para a fabricação de CIs, onde se necessita do chamado silício de grau eletrônico (**Si - GE**), com



pelo menos 99,999999% de pureza. Uma pequena parte deste **Si - GM** produzido é utilizado para a obtenção do **Si - GE** necessário à indústria de semicondutores. A maior parte é usada nas indústrias siderúrgica e química.

2.1.3 - Obtenção do Silício de Grau Eletrônico

O silício pode ser usado em microeletrônica porque tem propriedades como semicondutor extrínseco, as quais são definidas pela qualidade e pela quantidade de impurezas presentes no cristal. Semicondutores extrínsecos, em contraposição aos intrínsecos (cristais puros de determinados elementos químicos), são aqueles cujas propriedades semicondutoras são realçadas por impurezas dopantes combinadas com o cristal intrínseco. A construção de um CI consiste na introdução de certas impurezas no **Si - GE**, de forma controlada, por meio de vários processos.

Os elementos do grupo V-A, da tabela periódica, podem substituir átomos da rede cristalina do Si, doando-lhe um elétron (o Si possui quatro elétrons na camada de valência e os elementos do grupo V-A possuem cinco elétrons nesta camada). A partir de uma certa concentração destas impurezas, os mecanismos de condução do cristal de Si passam a ser governados pelos elétrons doados à rede. Neste caso, diz-se que o cristal de silício é do tipo-N, pois passam a ter um excesso de cargas negativas (elétrons doados), que serão as principais responsáveis pela condução de energia no semicondutor. O elemento mais usado como doador de elétrons é o fósforo (P).

Já os elementos do grupo III-A, possuem apenas três elétrons na camada de valência. Quando esses elementos substituem o Si na rede cristalina, provocam a falta de um elétron, o que equivale à criação de uma lacuna eletrônica na rede. Estas lacunas atuam como se fossem portadores de carga positiva. Portanto, aceitadores de elétrons. O Si dopado desta forma é dito silício tipo-P. Devido a certas características eletro-químicas inadequadas dos elementos deste grupo, o Boro (B) é praticamente a única escolha possível para utilização como impureza aceitadora no Si.

As técnicas de purificação direta não são capazes de produzir **Si - GE** para CIs. As técnicas que tiveram sucesso são as de purificação indireta, cuja diretriz básica é a seguinte: a partir do **Si - GM** produz-se um composto com um elemento intermediário, o qual deve ser purificado até atingir o grau eletrônico exigido; a partir deste composto purificado, faz-se a reação reversa para então obter o **Si - GE**.

O **Si - GE** policristalino obtido do processo de purificação servirá de base para a produção dos semicondutores que alimentarão a indústria de microeletrônica, viabilizando a fabricação de CIs. Contudo, este silício ainda não pode ser utilizado diretamente na confecção de circuitos, uma vez que sua estrutura atômica não é adequada para tanto. Antes deve-se submetê-lo a um processo de cristalização controlada, através do qual se obtém um cilindro de cristal de **Si - GE** com a estrutura atômica devidamente arranjada.



2.1.4 - Obtenção de Monocristais de Si - GE

Existem basicamente duas técnicas para o crescimento de cristais de silício, quais sejam: a de **Czochralski** (CZ) e a de **Fusão Zonal Flutuante** (FZ).

A técnica FZ é responsável por 10% da produção de monocristais de Si, enquanto a técnica CZ é representada por 90% da produção. [1]

O processo de crescimento de monocristais, utilizando-se o método CZ, inicia-se a partir de uma semente de cristal com a orientação cristalográfica apropriada, esta é parcialmente imersa no Si-GE derretido. A temperatura é reduzida até que o Si-GE derretido comece a solidificar em torno da semente, a qual é então lentamente afastada da superfície do Si-GE derretido. Escolhendo-se adequadamente a temperatura do processo e a velocidade de retirada da semente, a interface sólida-líquida permanece próxima à superfície da massa derretida e puxa-se dela um longo bastão de silício monocristalino. Para prevenir a oxidação do Si, este processo é conduzido em uma atmosfera inerte.

Portanto, o resultado destas técnicas, CZ e FZ, são bastões cilíndricos de Si monocristalino com cerca de um metro de comprimento e cujo diâmetro varia de acordo com a aplicação a que se destina.

Acontece que para ser utilizado na fabricação de CIs, o bastão monocristalino de **Si - GE** deve ser preparado ainda mais. Os CIs têm componentes com dimensões muito reduzidas, da ordem de alguns micra (menores, portanto, do que um grão de areia ou uma bactéria). Por conseguinte, as lâminas usadas em sua fabricação devem ter algumas características especiais, como veremos a seguir.

2.1.5 - Preparação das Lâminas de Si - GE

O processo de preparação das lâminas de **Si - GE**, a serem usadas na indústria de microeletrônica, envolve a **usinagem** e o **corte** do bastão de monocristal produzido com os métodos CZ ou FZ.

Como os equipamentos usados na fabricação dos CIs obedecem uma padronização, a primeira etapa desta preparação (usinagem), consiste no ajuste do diâmetro do bastão, para reduzi-lo aos padrões industriais. Este ajuste de diâmetro é feito através da retífica do bastão com um copo abrasivo de diamante. A seguir, com o mesmo copo de diamante, fazem-se chanfros no bastão retificado, para sua identificação.

Uma vez obtido um bastão de silício monocristalino, com as dimensões adequadas e perfeitamente identificado, passa-se à etapa seguinte que é o fatiamento (corte) deste bastão, para a obtenção das lâminas de Si que servirão para a fabricação dos mais variados circuitos integrados. Este fatiamento é feito por meio de uma serra circular de aço contendo, no centro, como instrumento de corte, um disco de diamante.



O bastão de silício é fixado a um suporte de cerâmica. Enquanto o disco gira a uma velocidade de aproximadamente 2000 rpm, o corte é feito pela translação da barra de silício a uma velocidade aproximada de 0,5 mm/s. O sistema é refrigerado a água deionizada. As lâminas resultantes têm uma espessura aproximada de 700 μ m.

A produção destas lâminas provoca uma perda de cerca de 30% do material da barra inicial de silício monocristalino, sob a forma de pó de serra.

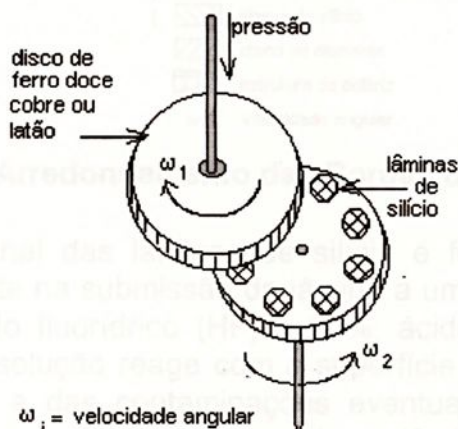


Figura 11 - Mecanismo de Lapidação das Lâminas de Si

Obtidas as lâminas de silício, deve-se fazer a lapidação das mesmas para tornar suas superfícies suficientemente paralelas. Isto é feito por meio de um equipamento com dois pratos giratórios, como ilustrado na figura 11. No prato inferior são fixadas as lâminas, aplicando-se uma cera em sua base inferior. Sobre as lâminas despeja-se uma solução abrasiva. Sobre esta montagem, faz-se girar outro disco, de ferro doce, cobre ou latão, sob pressão. Os discos inferior e superior giram com velocidades angulares diferentes e com sentidos de rotação opostos.

Nesta fase, normalmente, desbasta-se cerca de 20 mm a 50 mm de cada superfície, obtendo-se um paralelismo de aproximadamente 10 μ m entre as duas faces da lâmina.

Após lapidadas, sobram cantos vivos nas bordas das lâminas de Si. Estes cantos vivos podem causar pressões excessivas nas lâminas durante o manuseio, podendo provocar suas quebras ou dar origem a defeitos de deslocação da estrutura cristalina. Desta forma, é necessário que se faça o arredondamento das bordas das lâminas, o que é conseguido por meio de uma politriz, como ilustrado na figura 12. Após esta etapa, as lâminas estão prontas para o polimento final e a limpeza, como veremos a seguir.

2.1.6 - Epitaxia das Lâminas

Logo após obtidas as lâminas de silício monocristalino que serão usadas na fabricação dos chips, resta uma etapa na preparação destas lâminas antes de se iniciar o processo de sua transformação em chips. A



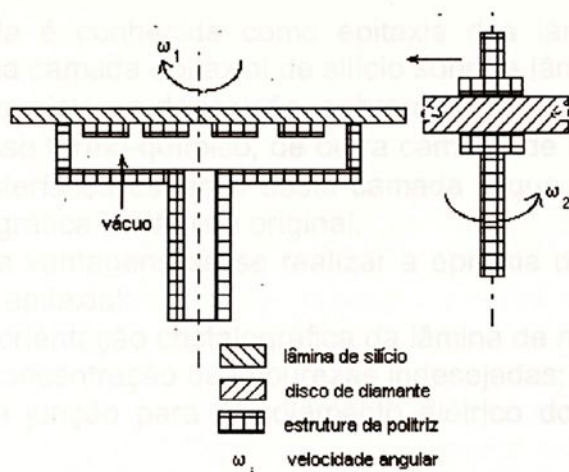


Figura 12 - Arredondamento das Bordas da Lâmina de Si

O polimento final das lâminas de silício é feito em duas etapas. A primeira etapa consiste na submissão da lâmina a um tratamento químico com uma solução de ácido fluorídrico (HF) a 20%, ácido nítrico a 45% e ácido acético a 35%. Esta solução reage com a superfície da lâmina, eliminando a maioria dos defeitos e das contaminações eventualmente introduzidos nos processos de corte e lapidação. A seguir, a lâmina é lavada com água deionizada em abundância, para retirar os resíduos desta reação. A segunda etapa consiste num polimento mecânico.

Após o polimento, deve-se proceder à limpeza final e à secagem das lâminas.

Em todo este processamento, bem como em todos os estágios da fabricação de CIs, devem-se usar compostos químicos de grau eletrônico. Por exemplo, a água deve ser super-deionizada, isenta de bactérias e de partículas suspensas. Esta água é obtida por processos osmóticos de retenção de íons. Um processo osmótico é aquele em que há uma disseminação espontânea de uma substância no interior de outra, através de uma membrana, resultando numa solução verdadeira entre estas duas substâncias[4]. A filtragem do ar ambiente também é de grande importância, para evitar a captação de pó pela lâmina, pois a lavagem lhe induz cargas eletrostáticas. Considerando-se as dimensões dos dispositivos nos CIs (atualmente fabricam-se circuitos cujas dimensões mínimas são inferiores a um micron), uma partícula de poeira pode arruinar completamente o circuito. Desta forma deve-se trabalhar em ambientes classe 100, no pior dos casos, o que significa que o ar deve ter menos de 1000 partículas suspensas por metro cúbico, as quais devem ser menores do que 0,5 mm.

2.1.6 - Epitaxia das Lâminas

Uma vez obtidas as lâminas de silício monocristalino que serão usadas na fabricação dos mais diversos CIs, resta uma etapa na preparação destas lâminas, antes de se iniciar o processo de sua transformação em *chips*. A



etapa aqui referida é conhecida como epitaxia das lâminas, ou etapa de crescimento de uma camada epitaxial de silício sobre a lâmina.

A epitaxia consiste na deposição, sobre a lâmina monocristalina de **Si - GE**, por um processo termo-químico, de outra camada de silício, com cerca de 25 μm . Uma característica especial desta camada é que ela segue a mesma orientação cristalográfica da lâmina original.

As principais vantagens de se realizar a epitaxia das lâminas originais são que a camada epitaxial:

- mantém a orientação cristalográfica da lâmina de monocristal;
- diminui a concentração de impurezas indesejadas; e
- forma uma junção para o isolamento elétrico dos circuitos a serem fabricados.

O primeiro item é importante pois é um dos principais objetivos de todo o processamento descrito nas cinco seções anteriores, deste capítulo. Sem esta perfeita orientação da estrutura cristalina do silício, não seria possível construir os dispositivos de microeletrônica utilizados atualmente.

O segundo item é importante porque, embora o processo de Czochralsky resulte em barras de **Si - GE** com a estrutura cristalina bastante perfeita, o processo não permite um controle adequado da concentração de impurezas. Como o crescimento da camada epitaxial é feito sob condições atmosféricas muito melhor controladas, a concentração de todas as substâncias desta nova camada é muito mais precisa, o que é de extrema importância para a realização dos CIs.

É importante observar que as lâminas de Si monocristalino saem do processo de produção com determinada dopagem inicial. Sobre cada uma dessas lâminas é acrescida uma camada de silício, com dopagem oposta à da lâmina original. Isto significa que, sobre uma lâmina do tipo P acrescenta-se uma camada epitaxial do tipo N, ou vice-versa. Com isto, tem-se camadas com cargas opostas, as quais devidamente polarizadas, e até mesmo por causa das características peculiares dos semicondutores, permitem que se crie uma região de espaço de cargas com uma resistividade tão alta que impeça a passagem de corrente elétrica, portanto, isolando o circuito a ser construído de seu substrato, que é a lâmina de monocristal.

2.2 - Processos de Fabricação dos Circuitos Integrados [12]

2.2.1 - Introdução

Após a preparação das lâminas de silício monocristalino, pode-se iniciar a fabricação dos mais variados CIs. Esta manufatura consiste na combinação de uma série de processos. Os sete mais importantes serão introduzidos nesta seção. São eles: oxidação, difusão, implantação de íons, deposição de vapores químicos, metalização, fotolitografia e encapsulamento.



Oxidação

A oxidação é o processo químico de reação do silício com o oxigênio para formar dióxido de silício. Este óxido é usado como isolante elétrico, protetor da lâmina de silício contra o ataque de impurezas indesejadas (contaminações), e servindo como máscara para a introdução controlada de dopantes na lâmina, durante as diversas etapas de fabricação.

Para acelerar esta reação de oxidação, as lâminas são aquecidas a temperaturas entre 1000°C e 1200°C.

Quando os elementos oxidantes são apresentados na forma de gases de alta pureza, tem-se a formação de óxido seco. Quando, ao contrário, a fonte de oxigênio é vapor d'água, forma-se óxido úmido. Ambos podem ser usados na fabricação de CIs. Porém, enquanto o óxido úmido forma-se mais rapidamente, as características elétricas do óxido seco são melhores. Por isso, geralmente o oxigênio gasoso é preferido, como fonte para a oxidação.

Difusão

O processo de difusão é aquele no qual átomos se movem na rede cristalina. Este processo permite a introdução de átomos de impurezas dopantes para alterar as características elétricas do silício.

Este processo é feito em duas etapas, a temperaturas entre 900°C e 1200°C.

A primeira etapa consiste na deposição dos átomos, de algum dopante, sobre a superfície desejada. Nesta etapa os átomos depositados ficam concentrados bem junto à superfície da lâmina. A segunda etapa consiste em fazê-los moverem-se pela estrutura cristalina, para que se disponham homogeneamente em profundidades maiores. Após a difusão, a lâmina é resfriada à temperatura ambiente, congelando as impurezas na rede cristalina do silício.

Dependendo da concentração de dopantes, pode-se ter um semiconductor fracamente dopado, com maior resistividade, usado nas partes dos elementos de circuitos, ou um semiconductor fortemente dopado, com baixa resistividade, usado como um condutor, por exemplo.

Implantação de íons

O processo de implantação de íons consiste na introdução de impurezas, na lâmina de silício, à temperatura ambiente. Isto é feito com a utilização de um equipamento que acelera íons da impureza desejada, por meio de um campo elétrico, e joga-os contra a superfície da lâmina. A profundidade de penetração é controlada pela intensidade do campo acelerador e a quantidade de íons implantados pela concentração do fluxo de íons, ou seja, pela corrente do feixe iônico. O resultado deste processo é mais preciso e mais reproduzível do que o da difusão. Contudo, também é um processo mais caro e, geralmente, requer um processo posterior de cozimento da lâmina, para a difusão dos íons pela estrutura cristalina. A implantação de íons é usada sempre que se necessita de concentrações mais bem controladas e de perfis mais homogêneos.



Deposição de vapores químicos dos Circuitos Integrados

O processo de deposição de vapores químicos, mais conhecido por sua sigla CVD (do inglês *chemical vapor deposition*), é a reação de gases ou vapores, para a formação de sólidos na superfície da lâmina. Este processo é semelhante ao de crescimento da camada epitaxial. Porém, neste caso, é usado para a obtenção de silício policristalino (poli-Si). A temperatura empregada na reação é inferior a 500°C. Normalmente, as camadas de poli-Si são fortemente dopadas, para formar regiões de alta condutividade usadas na interconexão dos dispositivos do CI.

Metalização

O processo de metalização consiste na deposição de um metal, geralmente o alumínio, para interligar os componentes que formarão o circuito integrado desejado. A metalização é conseguida pelo aquecimento do alumínio, no vácuo, até sua evaporação. O alumínio evaporado deposita-se sobre a superfície da lâmina. Após esta etapa de deposição, o alumínio é removido das partes indesejadas, por um processo de corrosão seletiva.

Fotolitografia

O processo de fotolitografia consiste na reprodução, sobre a superfície da lâmina de silício, das formas geométricas definidas pelo projetista do circuito, para representarem as diversas partes dos dispositivos do CI. Esta reprodução é feita fotograficamente. A primeira etapa desta reprodução consiste na cobertura de toda a superfície da lâmina com uma resina fotosensível. A seguir, esta resina é exposta à luz, através de uma película fotográfica. Esta película é simplesmente um negativo fotográfico de uma das partes dos dispositivos que devem ser construídos. A resina é então revelada, reproduzindo a película fotográfica, sobre a lâmina de silício. A parte da resina que ficou exposta à luz torna-se resistente à corrosão por um determinado tipo de agentes químicos, particularmente aqueles utilizados para corroer o SiO_2 e o Al. Desta forma, podem-se abrir janelas na camada de dióxido de silício, as quais são usadas, principalmente, nos processos de difusão.

Encapsulamento

Após processada, a lâmina de silício contém algo em torno de 100 a 1000 circuitos, ou *chips*. Cada *chip* tem entre 10 e dez milhões de transistores. Cada *chip* é retangular, com seus lados medindo entre 1 mm e 10 mm. Cada *chip* é testado, ainda na lâmina, por um equipamento automático de testes e os circuitos ruins são marcados com um pinga de tinta vermelha, para posterior descarte. Os circuitos são então separados e os bons são montados em suportes. A ligação das almofadas metálicas dos *chips* com os respectivos pinos do suporte é feita por meio de fios de ouro muito finos. Finalmente, este suporte é selado, em atmosfera inerte, ou no vácuo, antes de ser comercializado.



2.2.2 - Princípio de Funcionamento dos Circuitos Integrados

O mecanismo básico de funcionamento dos circuitos integrados pode ser entendido pelo estudo do comportamento de uma junção p-n. A junção p-n é o principal componente tanto dos circuitos bipolares quanto dos de efeito de campo. É com junções p-n que se formam diodos, transistores, resistores, capacitores e, até mesmo, as regiões de isolamento dos circuitos integrados. Por isto, a compreensão dos mecanismos físicos de operação de junções p-n é essencial para o entendimento da operação de todos os circuitos integrados.

A figura 13 mostra um esquema simplificado de uma junção p-n. As junções são formadas pela justaposição de uma porção de silício dopada positivamente a outra porção dopada negativamente. Isto é obtido, durante a fabricação de CIs, por exemplo, pela difusão consecutiva de uma camada de boro e outra de fósforo.

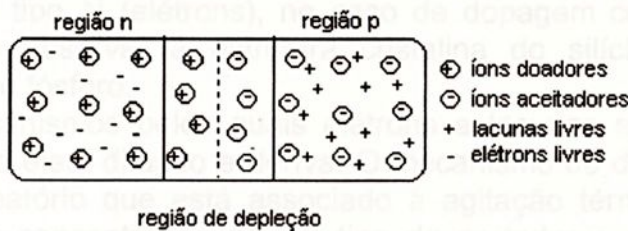


Figura 13 - Visão simplificada de uma junção p-n

O silício puro (intrínseco), apresenta em sua estrutura cristalina ligações covalentes entre seus átomos. Estas ligações são estáveis, em sua maioria, mas à temperatura ambiente algumas dessas ligações podem se romper, liberando um elétron. Este elétron liberado passa então a ser um portador de cargas, ou seja, está disponível para conduzir corrente elétrica. Por outro lado, este elétron deixa um buraco, ou lacuna, no átomo que o liberou. Esta lacuna apresenta uma carga igual à do elétron livre, porém, como se trata da falta de um elétron e os elétrons têm carga negativa, ela comporta-se como uma carga positiva. A lacuna de um átomo pode atrair um elétron do átomo vizinho. Quando isso ocorre, a lacuna se recombina com o elétron atraído, neutralizando-se. Contudo, agora o átomo vizinho é quem porta uma lacuna. Este processo de recombinação pode se repetir com o átomo vizinho, com o vizinho do vizinho, e assim por diante. O efeito geral é que a lacuna se comporta como um portador de uma carga positiva. Desta forma, podemos entender que num cristal de silício temos dois tipos de portadores de carga: os elétrons (que carregam cargas negativas) e as lacunas (que carregam cargas positivas).

Quando se adiciona fósforo ao silício, temos a formação de uma junção p-n. Como mostrado na figura 14, na região dopada positivamente os portadores majoritários são as lacunas e na região dopada negativamente os portadores majoritários são os elétrons. Logo que a junção é formada, a alta concentração de elétrons na região N faz com que estes se difundam no sentido da região P. Analogamente, a maior concentração de lacunas na região P provoca a difusão destas para a região N. Na medida em que estes portadores atravessam o limite das duas regiões, eles tendem a se recombinar.



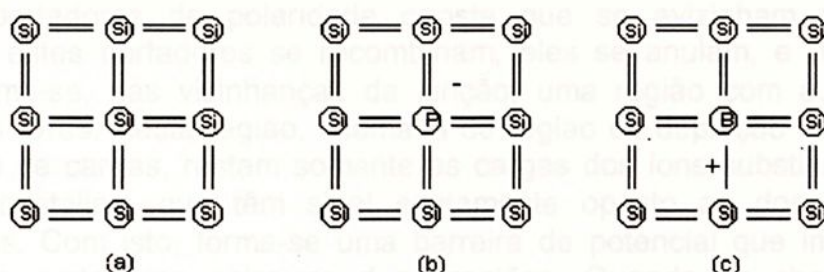


Figura 14 - Estrutura cristalina do silício

a - material intrínseco

b- após a adição de um átomo de fósforo

c- após a adição de um átomo de boro

O silício dopado, ou extrínseco, apresenta um excesso de portadores de carga. Tais portadores podem ser do tipo_P (lacunas), no caso de dopagem com o boro, ou do tipo_N (elétrons), no caso de dopagem com fósforo. Na figura 14, pode-se observar a estrutura cristalina do silício, quando da dopagem com boro e fósforo.

Há dois mecanismos pelos quais elétrons e lacunas se movem num cristal de silício, são eles: difusão e deriva. O mecanismo de difusão consiste num movimento aleatório que está associado à agitação térmica no cristal. Contudo, quando a concentração de um tipo de portadores se torna maior numa região do cristal, estes portadores tendem a se difundir, ou seja, se mover, da região de alta concentração para a região de concentração mais baixa. Este movimento de portadores é o que se chama de corrente de difusão, em semicondutores. O mecanismo de deriva ocorre quando um campo elétrico é aplicado ao cristal. Neste caso, os portadores são acelerados pelo campo elétrico e adquirem uma componente de velocidade que é superposta ao seu movimento térmico, resultando numa determinada corrente de deriva.

No silício intrínseco, as concentrações de lacunas e elétrons livres, gerados por ionização térmica, são iguais. No caso de semicondutores dopados, provoca-se intencionalmente o surgimento de regiões com uma maioria de portadores de um determinado tipo (elétrons ou lacunas). No caso de um semicondutor dopado com fósforo, por exemplo, a maioria dos portadores são do tipo-N (elétrons), embora haja uma minoria de portadores do tipo-P. No caso de dopagem com boro, os portadores majoritários são do tipo-P (lacunas), havendo, entretanto, uma quantidade minoritária de portadores do tipo-N.

Quando se tem uma peça de semicondutor uniformemente dopada, com um determinado tipo de portador, tudo se passa como no caso intrínseco. Por outro lado, quando se une, numa mesma peça de semicondutor, uma região dopada positivamente e outra dopada negativamente, temos a formação de uma junção p-n. Como mostrado na figura 4, na região dopada positivamente os portadores majoritários são lacunas e na região dopada negativamente os portadores majoritários são elétrons. Logo que a junção é formada, a alta concentração de elétrons na região N faz com que estes se difundam no sentido da região P. Analogamente, a maior concentração de lacunas na região P provoca a difusão destas para a região N. Na medida em que estes portadores atravessam o limite das duas regiões, eles tendem a se recombinar

com os portadores de polaridade oposta que se avizinham da junção. Conforme estes portadores se recombinam, eles se anulam, e após algum tempo forma-se, nas vizinhanças da junção, uma região com ausência de portadores livres. Nesta região, chamada de região de depleção ou de região de espaço de cargas, restam somente as cargas dos íons substitucionais da estrutura cristalina, que têm sinal exatamente oposto ao dos portadores majoritários. Com isto, forma-se uma barreira de potencial que impede esta difusão de portadores entre as duas regiões. Quando se chega a este equilíbrio tem-se, finalmente, formada a junção p-n.

Se a junção p-n for polarizada diretamente, como mostra a figura 15-a, a tensão aplicada forçará os elétrons da região N fluírem para a região P e as lacunas desta região moverem-se no sentido daquela. Como, neste caso, os portadores de carga são majoritários, um pequeno valor de tensão é suficiente para produzir uma corrente razoavelmente elevada.

Se a junção p-n for polarizada reversamente, como mostra a figura 15-b, a tensão aplicada forçará os elétrons da região P se dirigirem para a região N e as lacunas desta serão empurradas para aquela. Como, neste caso, tratam-se de portadores minoritários, o número de cargas em movimento é muito pequeno e, portanto, o fluxo de corrente é bastante reduzido, mesmo para tensões bem elevadas. Entretanto, chegando-se a uma determinada tensão, chamada de tensão de ruptura, ocorre o fenômeno de avalanche. Este fenômeno consiste na aquisição, por parte dos portadores minoritários, duma grande quantidade de energia.

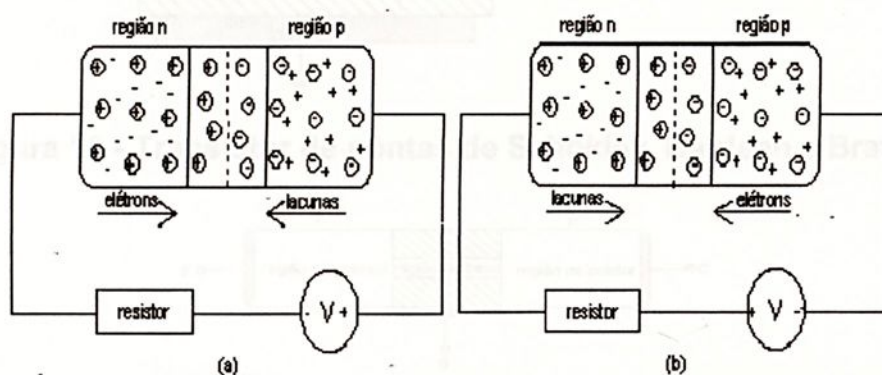


Figura 15 - Polarização de uma junção p-n (a) direta; (b) reversa

Após a apresentação do funcionamento de uma junção p-n, pode-se descrever as duas principais tecnologias de fabricação de circuitos integrados. Este estudo se restringirá ao funcionamento dos transistores bipolares e de efeito de campo. Estes dispositivos possuem três terminais e seu funcionamento, a grosso modo, consiste no controle da corrente num de seus terminais pela tensão aplicada entre os outros dois terminais. Estes dispositivos têm uma ampla gama de aplicações. Particularmente, como é possível usar o sinal de controle destes dispositivos, para fazer com que a corrente no terminal controlado varie rapidamente de zero até um valor alto, eles podem ser utilizados como uma chave eletrônica. Estas chaves são os elementos fundamentais para a implementação de sistemas digitais.

2.2.3 - Tecnologia Bipolar

Inventado em 1947 por Schokley, Bardeen e Brattain (figura 16), o transistor bipolar foi o responsável pela revolução eletrônica nos anos cinquenta. Nestes dispositivos a corrente é conduzida tanto por elétrons quanto por lacunas, donde vem sua denominação de bipolar. Fundamentalmente, este dispositivo consiste em duas junções p-n, unidas costa-a-costa, como mostra a figura 17. Na verdade, são três regiões dopadas, alternadamente, com impurezas doadoras ou aceitadoras, formando dois tipos duais de transistores, quais sejam: o tipo p-n-p e o tipo n-p-n. Os três terminais destes dispositivos são chamados de: emissor, coletor e base. A região central é a base do transistor. Cada par de regiões forma um diodo, um deles chamado de junção base-emissor (JBE), o outro de junção base-coletor (JBC). Dependendo da maneira como estas junções forem polarizadas, o transistor pode operar de modo diferente.

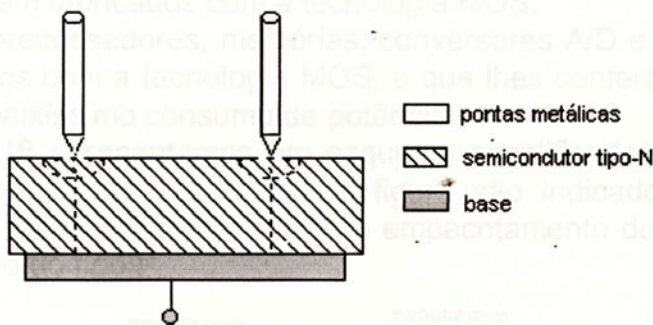


Figura 16 - Transistor de pontas de Schokley, Bardeen e Brattai

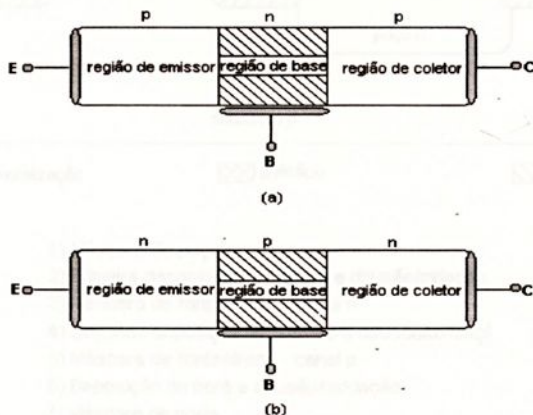


Figura 17 - Transistor (a) tipo p-n-p e (b) n-p-n

2.2.4 - Tecnologia MOS

Os transistores bipolares e os transistores MOS têm a corrente entre dois de seus terminais controlada pela tensão no terceiro terminal, mas ao

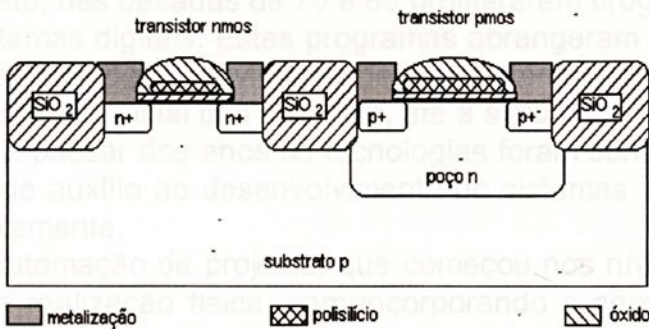


contrário dos bipolares, cujo controle é feito pela corrente no terminal de base, nos transistores MOS este controle se dá pela ação de um campo elétrico imposto ao terminal controlador. Por isso, estes dispositivos são chamados de transistores de efeito de campo (TECs). Neste caso, a corrente é transportada, basicamente, por apenas um tipo de portadores (elétrons ou lacunas), sendo estes dispositivos também ditos transistores unipolares.

Conceitualmente, os TECs são conhecidos desde a década de trinta. Porém, só se tornaram factíveis nos anos sessenta. No final da década de 70 surgiu um tipo particular de TECs: os transistores de efeito de campo cujo terminal controlador é construído como um sanduíche de metal, óxido e semiconductor (MOS). Estes transistores tornaram-se bastante populares, rapidamente, principalmente porque ocupam uma área de silício bem menor, consomem bem menos potência e passam por um processo de fabricação bem mais simples do que os transistores bipolares. Além disto, funções digitais podem ser implementadas usando exclusivamente transistores MOS, também representando um ganho de área, por dispensar a utilização de resistores. Estas razões fizeram com que a grande maioria dos circuitos integrados VLSI e ULSI atuais sejam fabricados com a tecnologia MOS.

Os microprocessadores, memórias, conversores A/D e D/A, etc... , são circuitos projetados com a tecnologia MOS, o que lhes confere alta densidade de integração e baixíssimo consumo de potência.

Na figura 18 apresentamos um esquema simplificado do processo de fabricação de um circuito MOS. Nesta figura são indicados os principais passos, desde a difusão do poço N, até o empacotamento do *chip*, sobre um substrato de silício do tipo P.



- 1) Máscara do poço p
- 2) Primeira deposição de fósforo e difusão/oxidação
- 3) Máscara de fonte/dreno - canal n
- 4) Segunda deposição de fósforo e difusão/oxidação
- 5) Máscara de fonte/dreno - canal p
- 6) Deposição de boro e difusão/oxidação
- 7) Máscara de porta
- 8) Oxidação das portas e máscara de poli-silício
- 9) Deposição de poli-silício
- 10) Máscara de porta
- 11) CVD das portas e máscara de contatos
- 12) Máscara de metalização
- 13) Metalização
- 14) Vitrificação
- 15) Encapsulamento

Figura 18 - Seqüência de Fabricação de Transistores com a Tecnologia MOS



3 - Projeto de Circuitos Integrados

3.1 - Introdução [19]

Durante os anos 60, o projeto de circuitos integrados era praticamente manual. O projetista elaborava um esquema de seu circuito com o auxílio de lápis e papel. Após o projeto do circuito, um especialista em *layout* desenhava as máscaras necessárias à integração, as quais eram transferidas para *rubylith* e cuidadosamente inspecionadas a olho nu, antes de serem reduzidas, por processo fotográfico, às dimensões adequadas para a fabricação. Na seqüência, eram feitos testes exaustivos no circuito fabricado de modo a determinar a correção do produto final.

Com o passar do tempo, o aumento da complexidade dos circuitos e a impossibilidade de acessar diretamente seus elementos internos, este processo tornou-se inviável. Os testes realizados nos circuitos após a sua fabricação, tornaram-se exaustivos e excessivamente grandes. Adicionalmente, os custos de projeto e de fabricação, tornaram-se cada vez maiores, obrigando os projetistas a uma avaliação prévia da correção dos circuitos. Ou seja, custava muito caro fabricar um circuito e ter que descartá-lo posteriormente devido a erros de projeto. Ademais, o tempo disponível entre a concepção de um sistema e sua colocação no mercado é uma variável crítica no seu desenvolvimento, uma vez que alguns dias de antecipação nas vendas de um concorrente, podem inviabilizar a comercialização de um produto, principalmente num mercado tão dinâmico quanto o de informática.

Pelo exposto, nas décadas de 70 e 80 proliferaram programas de auxílio ao projeto de sistemas digitais. Estes programas abrangeram as mais diversas etapas do processo de desenvolvimento desses sistemas, desde a simulação funcional da concepção inicial dos sistemas, até a simulação dos processos de fabricação. Com o passar dos anos as tecnologias foram sendo desenvolvidas e os programas de auxílio ao desenvolvimento de sistemas integráveis foram evoluindo conjuntamente.

Assim, a automação de projetos, que começou nos níveis de abstração bem próximos da realização física, vem incorporando e se confundindo com recursos e ferramentas para a validação de projeto em níveis cada vez mais altos de abstração. Expectativas de evolução dessa área para um futuro próximo se referem à disponibilidade de ambientes para projeto de sistemas digitais rápidos e confiáveis, que integrem todas as ferramentas de validação de projeto de uma forma confortável para a sua utilização e produzam sistemas digitais de alto desempenho, com baixo custo, que estejam funcional e eletricamente corretos.

3.2 - Níveis de Projeto [19]

O projeto de um sistema digital deve resultar em informações de duas naturezas, que são essenciais para o processo de fabricação: as informações **estruturais**, que descrevem os componentes utilizados e suas interconexões,



e as informações **geométricas**, que dão a localização exata dos componentes e rotas sobre um circuito impresso ou uma superfície de silício.

Para sistemas complexos, estas informações estruturais e geométricas não são uma tarefa trivial, sendo necessário trabalhar com diferentes níveis de projeto. Um nível de abstração mais elevado permite que o projetista manipule um número menor de componentes do que aquele realmente existente no sistema físico. Em segundo lugar, uma simplificação importante pode ser obtida se abstrairmos os detalhes estruturais e geométricos do sistema, realizando o projeto em termos do seu comportamento, que pode ser descrito através de um algoritmo, por exemplo. Este segundo tipo de abstração acrescenta informações de natureza **comportamental** ao processo de projeto.

O projeto de sistemas digitais é realizado, portanto, ao longo de 3 eixos ortogonais entre si, havendo sobre cada um deles diferentes níveis de projeto representando diferentes graus de abstração em relação ao sistema físico, o que pode ser visto no diagrama de Gajski-Kuhn, na figura 19.

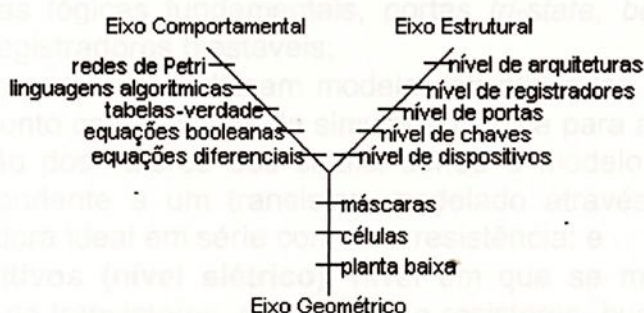


Figura 19 - Diagrama Y de Gajski-Kuhn

O centro dos eixos representa o sistema físico, enquanto os círculos concêntricos identificam pontos sobre os 3 eixos que correspondem a um mesmo nível de projeto. Pontos mais afastados da origem representam um maior grau de abstração.

Sobre o eixo geométrico, os níveis de projeto são muito dependentes da tecnologia empregada no desenvolvimento do sistema. Como estamos supondo o caso da implementação do sistema na forma de um circuito integrado, algumas fases do projeto da geometria do sistema, numa sequência decrescente de abstração, são:

- **planta baixa**, onde é feita uma primeira estimativa morfológica dos diversos blocos e de sua distribuição na superfície da área de silício disponível;
- **células**, onde resolvem-se as formas das células que comporão os diversos blocos do sistema; e
- **máscaras**, onde se projetam os conjuntos de polígonos que formarão cada uma das células a serem usadas no sistema, de acordo com as restrições impostas pelo projeto morfológico das células e pelas regras de projeto para o relacionamento das diferentes camadas utilizadas na tecnologia de fabricação subjacente (difusões, metalizações, implantações, etc).

Por sua vez, os pontos do eixo estrutural representam conjuntos diferentes de primitivas, a partir das quais pode-se compor um sistema.

Novamente, seguindo uma seqüência decrescente no nível de abstração, destacam-se os seguintes pontos neste eixo:

- **arquiteturas (nível de sistemas)**, onde busca-se verificar a concepção de grandes sistemas sem a necessidade de entrar em maiores detalhes do *hardware* subliminar, neste nível as primitivas básicas dos sistemas são processadores, memórias, controladores e periféricos;
- **registradores (nível RT)**, no qual procura-se descrever os componentes básicos do sistema digital do nível operacional, sendo que, neste caso, um registrador não implica apenas um registrador como entendido do ponto de vista da lógica seqüencial, mas também engloba todos os outros tipos de registradores, tais como biestáveis, registradores de deslocamento, contadores e multiplexadores, bem como unidades funcionais complexas tais como PLAs;
- **portas (nível lógico)**, onde vê-se o sistema como uma interconexão de portas lógicas fundamentais, portas *tri-state*, *buffers* e, quando muito, registradores biestáveis;
- **chaves**, no qual se utilizam modelos simplificados dos transistores, em conjunto com técnicas de simulação lógica para a propagação e a avaliação dos valores dos sinais, sendo o modelo básico utilizado correspondente a um transistor, modelado através de uma chave comutadora ideal em série com uma resistência; e
- **dispositivos (nível elétrico)**, nível em que se modela o sistema através de transistores, capacitores e resistores, buscando-se prever seu comportamento com o máximo de detalhes possível.

Finalmente, no eixo comportamental, há diversos formalismos para a representação funcional do sistema, dentre os quais destacam-se, em ordem decrescente de abstração, os seguintes pontos:

- **redes de Petri**, utilizadas na representação de fluxos de controle e de dados em sistemas descritos no nível de registradores ou de arquiteturas;
- **linguagens algorítmicas**, que permitem a descrição dos fluxos de controle de sistemas, bem como ações operacionais associadas a eles, desde o nível de portas até o de arquiteturas;
- tabelas-verdade e de transição de estados, respectivamente, utilizadas na descrição funcional de blocos combinacionais e na descrição de lógica seqüencial;
- **equações booleanas**, utilizadas para descrever a função de blocos de lógica combinacional; e
- **equações diferenciais**, utilizadas para a descrição matemática do comportamento dos sistemas no nível de dispositivos.

3.3 - Metodologias de Projeto [19]

Metodologias de projeto são seqüências de transformações que partem de uma descrição e/ou especificação inicial de um sistema digital para chegar a uma descrição final validada deste sistema, que é suficiente e necessária



para o processo de fabricação. Esta descrição final contém informações relativas aos eixos estrutural e geométrico de projeto, sendo feita a partir dos componentes primitivos do nível de abstração no qual é fabricado o sistema. Este *nível de fabricação* pode ser por exemplo, o das máscaras de fabricação, no caso de um circuito integrado.

As transformações podem ser de duas naturezas:

- **transformações de síntese** - geram uma descrição em um nível i , usando primitivas deste nível, de forma a atender uma especificação ou a manter uma equivalência com uma descrição de outro nível; as descrições inicial e final neste caso podem estar sobre o mesmo eixo de projeto ou sobre eixos diferentes;
- **transformações de validação** - levam uma descrição do estado não-validado para o estado validado, em função de requisitos de projeto que são próprios do nível de descrição em questão. Este requisitos são de natureza funcional ou se relacionam com restrições geométricas, de custo, velocidade, etc. No nível mais abstrato de projeto, estes requisitos correspondem à *especificação* inicial do sistema a ser projetado.

Uma metodologia de projeto particular pode empregar um processo *descendente* ou *ascendente* de concepção, ou ainda uma combinação de ambos. Um processo descendente de projeto é aquele no qual uma seqüência de transformações de síntese e de validação é utilizada, partindo de um nível i e chegando a um nível $i-j$ (processo de refinamento). No processo ascendente a seqüência de transformações de síntese e validação parte de um nível i para chegar a um nível $i+j$ (processo de composição). Em princípio, a cada transformação de síntese deve seguir uma transformação de validação, exceto se for utilizada síntese automática, o que pode ocorrer num processo descendente de projeto.

Usualmente, toda metodologia de projeto realiza uma combinação dos processos ascendentes e descendentes. O projeto é feito normalmente de forma descendente até que se atinja um nível N no qual as primitivas, sejam elas estruturais ou comportamentais, tenham uma implementação conhecida em termos de primitivas estruturais do nível de fabricação. Estas primitivas do nível N formam uma *biblioteca de funções* (ou células), e sua concepção é normalmente realizada de forma ascendente.

Transformações podem ser feitas *manual* ou *automaticamente* com o auxílio de *ferramentas de projeto*. Transformações de validação automática são realizadas por simuladores ou programas de verificação. Transformações de síntese manual são realizadas com o auxílio de compiladores de linguagens de descrição de *hardware* ou de editores gráficos, já que nos níveis de projeto menos abstratos descrições gráficas são preferíveis às textuais.



3.2 - Projeto Lógico

3.2.1 - Conceitos Introdutórios [5]

Quando um projetista realiza um projeto lógico este não precisa preocupar-se com detalhes físicos de implementação da máquina e assim mesmo é possível analisar diversos aspectos do funcionamento da máquina pronta, esta é uma grande contribuição do projeto no nível lógico. É possível representar uma máquina como uma caixa preta, que comporta-se de forma previsível e é representada por um diagrama de blocos. O comportamento de uma máquina de estados refere-se ao relacionamento das entradas e saídas para a máquina. Veja o esquema da figura 20.

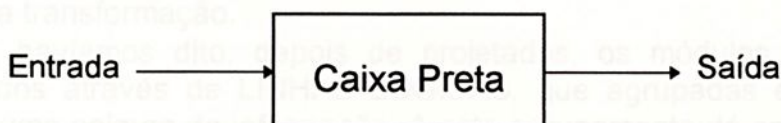


Figura 20 - Relacionamento da entradas/saídas com a máquina

A caixa preta é usada no projeto lógico para descrever o comportamento da máquina, esta baseada na *Álgebra de Boole*, dá a possibilidade de especificar este comportamento de forma exata e sem ambigüidades, principalmente devido à utilização de valores binários, 0 e 1.

O comportamento de uma caixa preta pode ser melhor especificado, quando a dividimos em **módulos**, de tamanho e complexidade menores. Assim, é possível realizar o projeto de cada módulo individualmente e ao final interconectá-los para que realizem, juntos, a tarefa designada pela caixa preta.

A esta divisão da caixa preta em módulos dá-se o nome de **partição funcional**.

Os três módulos principais são: MEMÓRIA, TRANSFORMAÇÃO e CONTROLE.

As **memórias** são módulos que transportam informação através do tempo, ou seja, elas armazenam informações, *bytes* ou palavras, compostas por valores lógicos (*bits*, 0 e 1). Cada palavra na memória é localizada numa posição descrita por um **endereço**.

Os principais tipos de memórias são:

- ROM (Read Only Memory), não permitem que o seu conteúdo seja alterado, ou seja, pode-se apenas acessá-las para leitura dos dados, os quais são armazenados durante a fabricação da memória.
- RAM (Read Access Memory) permitem que seu conteúdo seja alterado, ou seja, possibilita ao usuário modificar, inserir e apagar dados. Estas memórias são acessadas de forma periódica, de acordo com o relógio do sistema.
- Registradores, também permitem que seu conteúdo seja alterado, só que de forma imediata. É importante ressaltar que este tipo de memória trabalha junto aos processadores, logo seu acesso é muito

mais rápido que a RAM, embora sua capacidade de armazenamento seja menor.

Os módulos de **transformação** produzem um conjunto de saída para um conjunto de entradas, onde considera-se o relacionamento lógico entre as entradas e saídas no decorrer do tempo.

Alguns exemplos são:

- Multiplexadores - empreendem algumas saídas para um grande número de entradas.
- Demultiplexadores - convertem algumas entradas em muitas saídas.
- Somadores - produzem a soma de dois números.

Controle produz saídas que serão responsáveis pelo controle da memória e da transformação.

Como havíamos dito, depois de projetados, os módulos podem ser interconectados através de LINHAS LÓGICAS, que agrupadas em paralelo transportam uma palavra de informação. A este agrupamento dá-se o nome de BARRAMENTO.

Assim, temos que a ESTRUTURA DE BARRAMENTO descreve a interconexão lógica entre os grupos de módulos. Geralmente, esta estrutura é feita de maneira que com poucos barramentos é possível garantir velocidade e flexibilidade necessárias para atender os requisitos de comunicação entre os módulos.

3.2.2 - A Máquina de Estados [5]

Cada módulo de um projeto lógico pode ser representado por um modelo geral chamado MÁQUINA DE ESTADOS. Este modelo contém os elementos essenciais para descrever um módulo em termos de suas entradas, saídas e tempo.

A figura 21 mostra os três elementos essenciais em uma máquina de estados: FUNÇÃO DE PRÓXIMO ESTADO, ESTADO e FUNÇÃO DE SAÍDA.

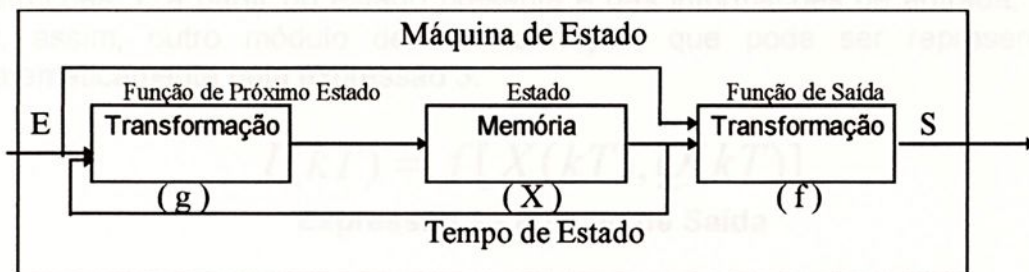


Figura 21 - Máquina de Estados

Na figura 21, temos uma entrada de dados, E, chamada qualificador, e uma saída de dados, S, chamada instruções.

O **estado** de uma máquina é a memória com capacidade suficiente para definir o comportamento futuro a partir do comportamento passado, ou seja, armazena informação suficiente para definir quais serão as futuras saídas de dados da máquina de estados e qual será o próximo estado. Por próximo estado entende-se, a nova informação que será responsável pelo comportamento da máquina em operações futuras. Geralmente, tais memórias são constituídas por circuitos biestáveis chamados FLIP-FLOPS.

Cada máquina de estados tem um próximo estado determinado por uma **função de próximo estado**. O TEMPO DE ESTADO (intervalo de tempo) é normalmente determinado por entradas periódicas de dados no registrador de estado. Assim, a cada intervalo de tempo, um novo estado (conjunto de informações na memória) substitui o estado anterior.

Normalmente, usa-se a seta ($\leftarrow$) para operações que não ocorrem imediatamente e o igual ($=$) para as imediatas.

A expressão 1 mostra o comportamento da função de próximo estado (g).

$$X((k+1)T) = g[X(kT), Q(kT)]$$

Expressão 1 - Função de Próximo Estado

A função de próximo estado, **g**, determinará um novo estado, dependente do estado presente $X(kT)$, e das entradas de dados ou qualificadores, $Q(kT)$. O tempo de estado é representado por T e k é uma contador inteiro, portanto, kT determina os intervalos de tempo em que o estado se altera.

Vale observar que a notação mais correta para a expressão 1 é a utilizada na expressão 2.

$$X \leftarrow [X, Q]$$

Expressão 2 - Função de Próximo Estado com Atraso

Na expressão 2, o operador de atraso ($\leftarrow$) indica que a mudança de estado se dá no próximo intervalo de tempo e que o próximo estado depende do estado presente e dos dados de entrada atuais.

A **função de saída (f)** gera um conjunto de saídas de dados ou instruções, **I**, a partir do estado presente e das informações de entrada, tem-se, assim, outro módulo de transformação, que pode ser representado matematicamente pela expressão 3.

$$I(kT) = f[X(kT), Q(kT)]$$

Expressão 3 - Função de Saída

Na expressão 3, observamos que os dados de saída são dependentes do estado presente, $X(kT)$, e das entradas atuais, $Q(kT)$. Também, vê-se que as saídas são imediatas pelo uso do igual ($=$).

A partir desta expressão, nota-se que uma máquina de estado trabalha ordenadamente com o tempo, assim, ela alcança a estabilidade durante cada



tempo de estado, kT . Produz-se, desta forma, o próximo estado e as saídas durante um período estável do tempo de estado.

3.2.3 - Descrição de Classes de Máquinas [5]

Existem 3 linguagens para descrever operações lógicas sobre os módulos de uma máquina de estado que são: TABELAS VERDADE, EXPRESSÕES BOOLEANAS e MAPAS DE KARNAUGH.

a) lógica e expressões booleanas

Geralmente, as informações dentro de sistemas digitais, como computadores ou equipamentos eletrônicos (vídeos, câmeras, etc.), são representadas por sinais binários quando do projeto lógico, porque o projeto se torna mais simples e seguro.

Assim, a teoria matemática utilizada no projeto lógico baseia-se na *Álgebra de Boole*, a qual utiliza valores binários, tais como, 0 e 1.

As operações mais comuns utilizadas na álgebra de Boole são o AND, OR e NOT. Há também operações mais complexas correspondentes à combinações destas operações básicas. Vejamos nas tabelas 1, 2 e 3 como funcionam estas operações básicas.

A	B	Or(+)
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 1

A	B	And(.)
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 2

A	Not(-)
0	1
1	0

Tabela 3

Equações booleanas são expressões obtidas pela da combinação de operações básicas, exibidas na tabela 1, 2 e 3, aplicadas a variáveis que assumem somente os valores 0 e 1.

O projetista utiliza estas equações para descrever o comportamento da máquina de estados e utiliza postulados e teoremas da Álgebra de Boole para manipular tais equações, a fim de que o projetista tenha condições de alterar e prever o comportamento futuro da máquina.

b) tabelas verdade

As tabelas verdade são representações de equações, mais precisamente, são tabelas que representam todos os possíveis valores de uma equação. A tabela 4 mostra a tabela verdade da expressão 4.



$$(A + B).C$$

Expressão 4 - Equação Booleana

A	B	C	(A+B).C
0	0	0	0
0	0	1	0
0	1	0	0
1	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0
1	1	1	1

Tabela 4 - Tabela Verdade da Expressão 4

c) mapas de karnaugh

Um Mapa de Karnaugh é uma maneira simplificada de representar uma tabela verdade.

A tabela 5, mostra o mapa de Karnaugh de três variáveis referente à tabela 4.

		AB			
		00	01	11	10
C	1	0	0	0	0
	0	0	1	1	1

Tabela 5 - Mapa de Karnaugh

O mapa de Karnaugh é composto por células, que são a intersecção de linhas com colunas. Margeando as linhas e as colunas colocam-se os nomes das variáveis e todos os possíveis valores para estas variáveis.

Em cada célula do mapa é colocado o valor que a expressão assume, com os correspondentes valores de entrada indicados nas margens horizontal e vertical da tabela. Para números diferentes de variáveis, tem-se tabelas diversas. Na figura 4 são mostradas as tabelas de karnaugh para 2, 3 e 4 variáveis.

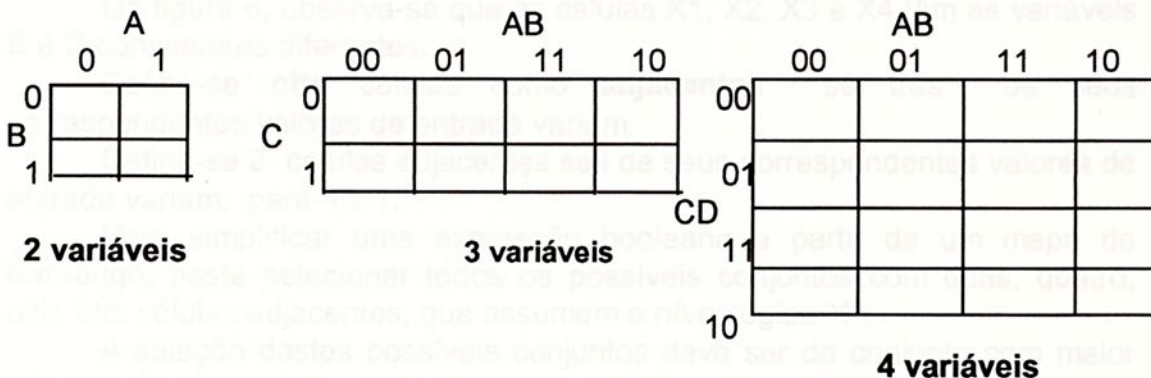


Figura 4 - Mapa de Karnaugh, para 2, 3 e 4 variáveis

Outra função associada aos mapas de Karnaugh é a simplificação de expressões booleanas, ou seja, transformar expressões complicadas em expressões mais simples que mantenham as mesmas características das originais. Assim, simplificando-se as expressões booleanas, a implementação física do projeto torna-se mais simples de ser realizada.

Antes de mostramos o método, vamos introduzir a noção de **adjacência**.

Define-se, duas células como **adjacentes** se apenas um dos correspondentes valores de entrada, indicados nas margens horizontal e vertical da tabela, varia (0 $\rightarrow$ 1, ou 1 $\rightarrow$ 0). A figura 5 mostra um exemplo de duas células adjacentes.

		AB			
		00	01	11	10
CD	00				
	01			X	Y
	11				
	10				

Figura 22- Duas células adjacentes

Da figura 22 observa-se que X é adjacente a Y pois, os valores correspondentes a X são 1101 e os correspondentes a Y são 1001, ou seja, X e Y diferem apenas na variável B.

Define-se quatro células como **adjacentes** se dois de seus correspondentes valores de entrada variam. A figura 6 mostra um exemplo de quatro células adjacentes.

		AB			
		00	01	11	10
CD	00			X1	X2
	01			X3	X4
	11				
	10				

Figura 23 - Quatro células adjacentes

Da figura 6, observa-se que as células X1, X2, X3 e X4 têm as variáveis B e D com valores diferentes.

Define-se **oito** células como **adjacentes** se três de seus correspondentes valores de entrada variam.

Define-se 2^i células adjacentes se i de seus correspondentes valores de entrada variam, para $i \geq 1$.

Para simplificar uma expressão booleana a partir de um mapa de Karnaugh, basta selecionar todos os possíveis conjuntos com duas, quatro, oito, etc. células adjacentes, que assumem o nível lógico "1".

A seleção destes possíveis conjuntos deve ser do conjunto com maior número de elementos, para o menor, onde a expressão resultante conteria apenas as entradas cujo valor permanece constante para todos os elementos do conjunto.

Para converter os conjuntos selecionados em expressões, deve-se fazer a conjunção das variáveis (A,B,C,...) ou de seus complementos (A^* , B^* , C^* , ...),



dependendo do valor que a variável assume. Adotando a variável A como exemplo, temos:

- a) A = 1, então usa-se o próprio A na expressão.
- b) A = 0, então usa-se A* na expressão.

Após a conversão dos conjunto em expressões, deve-se fazer a disjunção de todas as expressões resultantes dos conjuntos selecionados.

Exemplo: Utilizar Karnaugh sobre a expressão booleana, f, para simplificá-la.

$f = x1*x2*x3*x4* + x1x2x3*x4* + x1*x2x3*x4+ x1x2x3*x4+ x1x2x3x4*+ x1x2*x3x4*.$

O mapa de Karnaugh para f é mostrado na tabela 6:

		x1x2			
		00	01	11	10
x3x4	00	0	1	1	0
	01	0	1	1	0
	11	0	0	0	0
	10	0	0	1	1

Tabela 6 - Simplificação de expressão usando mapa de Karnaugh

Pelo exposto na tabela 6, pode-se ver claramente que foram selecionados dois conjuntos, um com quatro células e outro com duas células.

No conjunto com quatro células, observa-se que as variáveis x2 e x3 não variam, ou seja, $x2 = 1$ e $x3 = 0$ nas quatro células. Convertendo este conjunto em expressão, o resultado será $x2x3*$.

No conjunto com duas células, observa-se que as variáveis x1, x3 e x4 não variam, ou seja, $x1 = 1$, $x3 = 1$ e $x4 = 0$, nas duas células. A expressão resultante para este conjunto será $x1x3x4*$.

Fazendo a disjunção dos conjuntos selecionados, teremos $x2x3* + x1x3x4*$, que é a uma expressão simplificada da expressão f.

Mapas de Karnaugh e tabelas verdade, são muito úteis a qualquer projetista, embora só seja aconselhável utilizar os mapas para um número reduzido de variáveis de entradas, pois mapas de Karnaugh se tornam mais confusos quando o número de variáveis aumentam.

3.4.1 - Circuitos Lógicos [17]

Como foi dito, uma máquina de estados é descrita por funções booleanas. Para cada uma das operações da Álgebra de Boole (And, Or, Not, etc.) existe um símbolo que representa estas operações. Assim, a partir das equações booleanas pode-se montar o circuito lógico referente a esta equação, utilizando-se os símbolos apresentados na figura 23:



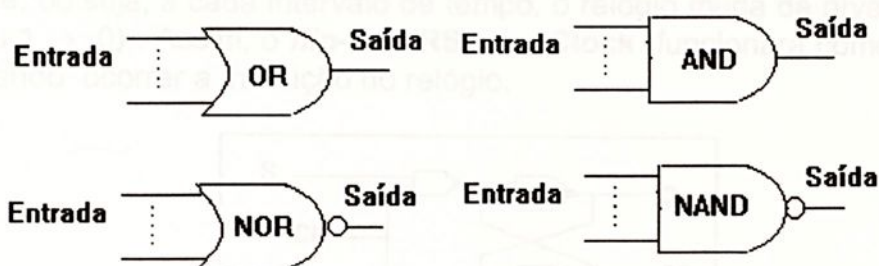


Figura 23 - Representação simbólica das operações lógicas

Para a simplificar o projeto lógico, podem-se utilizar alguns símbolos, símbolos estes representando um conjunto de portas lógicas que se repetem várias vezes dentro do circuito lógico. Iremos fazer uma menção aos mais utilizados.

a) Flip-Flops - são circuitos em que sua saída depende das entradas e do estado anterior. Logo, sua principal função é a de armazenar um *bit* de informação. Os *flip-flops* podem ser controlados por pulsos, de um relógio que alimenta todo o sistema, neste caso os *flip-flop* são chamados de *flip-flops* com **CLOCK**.

Nas figuras 24, 25, 26, 27 e 28 temos alguns exemplos de *flip-flops*, cada um com características diferenciadas, propiciando ao projetista escolher o mais adequado ao seu projeto.

O *flip-flop* RS, mostrado na figura 24, é o mais simples dos tipos de *flip-flops*. Os elementos que compõe o *flip-flop* são:

a) entradas R (*reset*) e S (*set*).

b) saídas Q e Q* (pôr definição Q* é a negação lógica do sinal Q).

Define-se ainda, Q_a como uma saída Q em um tempo anterior ao atualmente considerado.

Resumidamente, funcionam da seguinte forma:

- Quando as entradas $R = 0$ e $S = 0$, as saídas, $Q = Q_a$ e $Q^* = Q_a^*$.
- Quando $S = 0$ e $R = 1$, então $Q = 0$ e $Q^* = 1$.
- Quando $S = 1$ e $R = 0$, então $Q = 1$ e $Q^* = 0$
- Quando $S = 1$ e $R = 1$, a saída do *flip-flop* é imprevisível

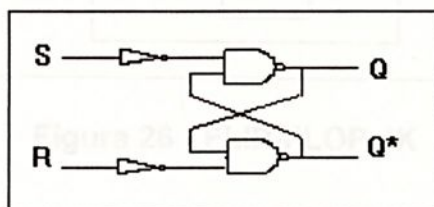


Figura 24 - FLIP-FLOP RS sem CLOCK

O *flip-flop* RS com *clock* mostrado na figura 25, é quase idêntico ao da figura 24, ou seja, duas entradas S e R, duas saídas Q e Q*, o estado anterior Q_a e adicionalmente um relógio, CK. O relógio funciona como um instrumento de

controle, ou seja, a cada intervalo de tempo, o relógio muda de nível lógico ($0 \rightarrow 1$ ou $1 \rightarrow 0$). Assim, o **flip-flop RS com Clock** funcionará como o *flip-flop RS* quando ocorrer a transição do relógio.

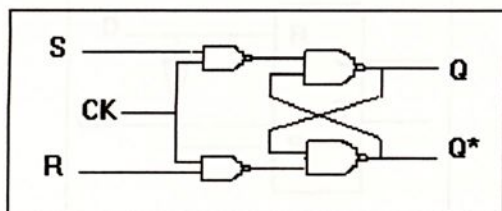


Figura 25 - FLIP-FLOP RS com CLOCK

O *flip-flop JK* é apresentado na figura 26. Ele é mais geral que os outros *flip-flops*, pois trabalha com as características já conhecidas do *flip-flop RS com Clock* e, adicionalmente, evita o problema das entradas $R = S = 1$, terem saída imprevisível.

O JK, é um *flip-flop* composto por um RS acrescido de duas entrada, J e K, respectivamente ligadas às entradas R e S através de duas portas AND. Estas portas tornam possível a previsão do valor lógico da saída Q quando as entradas R e S assumem, ambas, o valor lógico 1 ($R = S = 1$). O funcionamento de um *flip-flop JK* pode ser descrito da seguinte forma:

- J = 0 e K = 0 então $Q = Q_a$
- J = 0 e K = 1 então $Q = 0$
- J = 1 e K = 0 então $Q = 1$
- J = 1 e K = 1 então $Q = Q_a^*$

É importante considerar que estas saídas poderão ser alteradas se as entradas forem alteradas e também ocorrerem transições no sinal do relógio ($1 \rightarrow 0$ ou $0 \rightarrow 1$), caso contrário, o estado final permanecerá o mesmo.

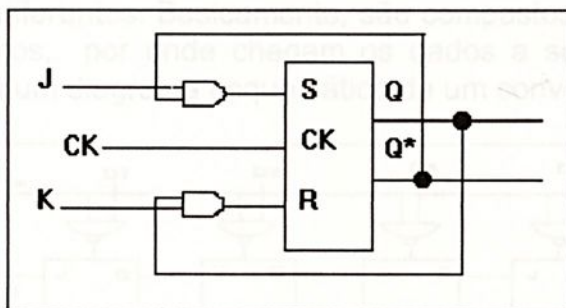


Figura 26 - FLIP-FLOP JK

A figura 27 mostra um tipo de *flip-flop* que é uma variação do RS, ou seja, no *flip-flop RS* conectou-se uma entrada D à entrada R, e a negação lógica de D, à S.

Resumido, este *flip-flop* trabalha da seguinte forma:

- Quando $D = 1$ então $Q = 1$.
- Quando $D = 0$ então $Q = 0$.

Q sinal responsável pela seleção das entradas do multiplexador a serem colocadas nas saídas é geralmente denominado **seleção**.

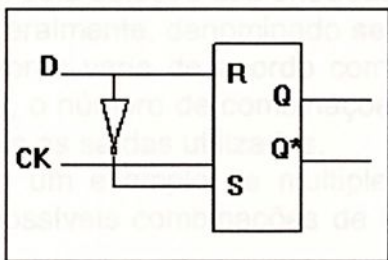


Figura 27 - FLIP-FLOP tipo D

O *flip-flop* da figura 28 é outra variação do *flip-flop* RS, este possui uma entrada T conectada a ambas as entradas R e S. O funcionamento deste *flip-flop*, também é controlado pelo *Clock*. Basicamente ele trabalha da seguinte forma:

- $T = 0$ então $Q = \underline{Q_a}$.
- $T = 1$ então $Q = \underline{Q_a}$.

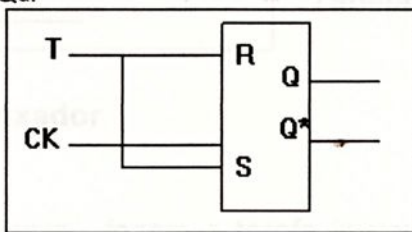


Figura 28 - FLIP-FLOP tipo T

b) Conversores - são circuitos lógicos responsáveis pela transformação de um conjunto de sinais (grupo de entradas A,B,C,...) em outro conjunto de sinais diferentes. Basicamente, são compostos de *flip-flops*, portas NAND e barramentos, por onde chegam os dados a serem convertidos. A figura 29 apresenta um diagrama esquemático de um conversor de sinais.

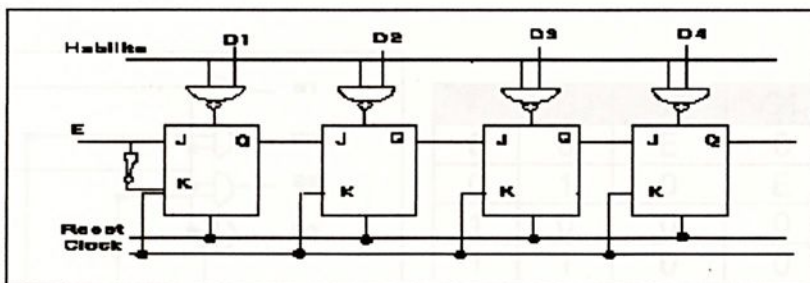


Figura 29 - Esquema de um conversor de sinais

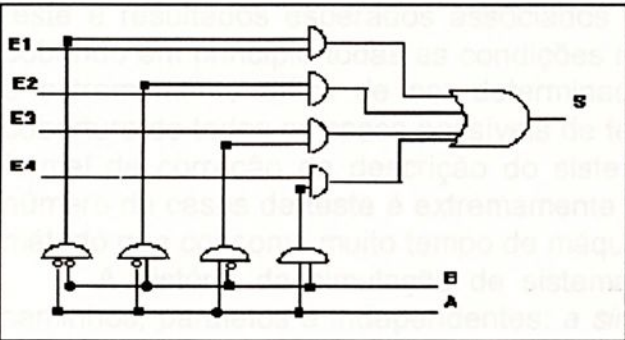
c) **Multiplexadores** - são circuitos que selecionam algumas das entradas paralelas de dados e as colocam em saídas, possivelmente

paralelas. É importante considerar que o número de entradas sempre é maior que o número de saídas, e que a saída pode ser única.

O sinal responsável pela seleção das entradas do multiplexador a serem colocadas nas saídas é, geralmente, denominado **seletor**.

O número de seletores varia de acordo com o número de entrada de dados, mais precisamente, o número de combinações atribuídas aos seletores irá selecionar qual ou quais as saídas utilizadas.

A figura 30 mostra um exemplo de multiplexador com 4 entradas, 1 saída e 2 seletores. As possíveis combinações de entradas com a saída são mostrados na tabela 7:



A	B	S
0	0	E1
0	1	E2
1	0	E3
1	1	E4

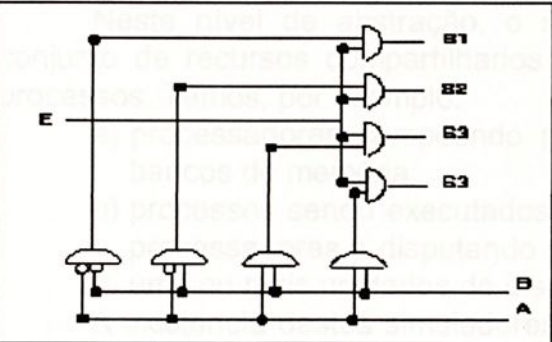
Tabela 7 - Combinações do Multiplexador

Figura 30 - Multiplexador

d) demultiplexadores - fazem a tarefa inversa ao multiplexador. Este circuito atribui as entradas de dados entre as possíveis saídas. O número de saídas é maior que o número de entradas, sendo possível existir uma única entrada.

A seleção de quais saídas serão utilizadas se dá através do **seletor**. O número de seletores é proporcional ao número de saídas possíveis.

O exemplo da figura 31 mostra um demultiplexador composto por 1 entrada, 4 saídas e 2 seletores, e a tabela 8 mostra as possíveis combinações de entradas e saídas.



A	B	S0	S1	S2	S3
0	0	E	0	0	0
0	1	0	E	0	0
1	0	0	0	E	0
1	1	0	0	0	E

Tabela 8 - Combinações do multiplexador

Figura 31 - Demultiplexador



4 - Simulação

4.1 - Visão Geral

Simulação é o processo de validação de um modelo de um sistema. Na simulação, um conjunto de vetores de teste é aplicado a um modelo do sistema, e os resultados são comparados com o resultado esperado. A simulação é o método mais tradicional de validação, mas também o que tem maiores limitações. Ela exige a determinação de um conjunto de vetores de teste e resultados esperados associados que seja mínimo mas significativo, cobrindo em princípio todas as condições reais possíveis de ocorrerem, o que é extremamente difícil de ser determinado. Não havendo esta certeza da cobertura de todos os casos possíveis de teste, a simulação não dá uma prova formal de correção da descrição do sistema. Além disso, de modo geral, o número de casos de teste é extremamente grande, o que faz da simulação um método que consome muito tempo de máquina.

A história da simulação de sistemas digitais apresenta dois grandes caminhos, paralelos e independentes: a *simulação de sistemas* e a *simulação lógica*. Além deles, a *simulação RT*, também está se tornando uma ferramenta convencional de projeto, enquanto a *simulação multi-nível* é o desenvolvimento mais recente.

4.2 - Os Tipos de Simuladores

a) *Simulação de Sistemas* [12]

O nível de sistemas é aquele no qual um sistema digital é visto como uma interconexão de processadores, memórias, canais de E/S, etc. Neste nível de abstração, o objetivo de um projeto é determinar o número de componentes de cada um dos tipos acima, a forma de interconexão entre eles, os algoritmos para arbitramento de conflitos de acesso a recursos compartilhados e parâmetros (velocidade, capacidade) que definem cada componente.

Neste nível de abstração, o sistema digital é representado por um conjunto de recursos compartilhados por cuja utilização competem diversos processos. Temos, por exemplo:

- a) processadores competindo por um único barramento de acesso a bancos de memória;
- b) processos sendo executados num mesmo processador ou em vários processadores e disputando o uso de canais de E/S para acessarem uma ou mais unidades de disco.

A existência destes simuladores, que suportam modelos adequados ao nível de sistemas e oferecem facilidades diversas, explica porque não foram desenvolvidas técnicas específicas de simulação de sistemas digitais neste nível de abstração.



b) Simulação Lógica [12]

Os simuladores lógicos se aplicam a circuitos descritos exclusivamente com portas lógicas elementares (E, OU, inversor, etc.), no entanto, existem alguns simuladores ditos lógicos que incluem também *flip-flops* e/ou transistores modelados como chaves. O principal objetivo deste tipo de simulador é obter, a partir de determinadas seqüências de sinais aplicadas às suas entradas, valores lógicos de saídas do circuito em função do tempo.

A simulação lógica pode ser usada com um de dois objetivos: *verificação lógica* e *verificação temporal*. Na verificação lógica, interessa unicamente ao projetista a correção das equações lógicas, que descrevem o comportamento do circuito, sem qualquer consideração de tempo. Este tipo de simulação lógica é útil para a simulação de circuitos síncronos. Já a verificação temporal, que considera atrasos de propagação das portas, é útil para a validação de circuitos assíncronos.

Os simuladores lógicos podem ser divididos em duas categorias, de acordo com os atrasos dos sinais no circuito, são elas, os simuladores de atrasos simples e os de atrasos múltiplos.

Os simuladores de atrasos simples podem ser subdivididos em simuladores de atrasos nulos, os quais não consideram os atrasos dos sinais nos circuitos, e simuladores de atrasos unitários, os quais consideram atrasos idênticos para todas as portas que compõem o circuito.

Existem dois tipos de simuladores de atrasos múltiplos, o de atrasos específicos e o de atrasos variáveis. Nos simuladores de atrasos específicos, os atrasos são considerados como assumindo um valor médio para todas as portas de um determinado tipo, porém este valor é diferente para portas distintas. Já nos simuladores de atrasos variáveis, são considerados valores distintos para cada uma das portas do circuito, podendo-se, inclusive, definir faixas de tempos de resposta para as portas, tempos de subida e descida dos sinais, etc.

Os primeiros simuladores, descritos na literatura na década de 60, eram de atrasos simples, eles serviam apenas para a verificação lógica (ou funcional) de circuitos lógicos combinatórios ou seqüenciais síncronos. Por verificação funcional entende-se a análise do circuito projetado de forma a verificar a correção da implementação da função lógica usada na sua especificação.

Embora a simulação lógica tenha sua utilidade nas fases iniciais de projeto, os projetistas necessitam de um *software* que lhe permita verificar o circuito projetado sob os diversos aspectos que envolvem o processamento dos sinais. Para tais tarefas são necessário simuladores de atrasos múltiplos. Tais simuladores devem ser capazes de verificar o projeto, detectando anomalias no chaveamento nas entradas e saídas das diversas portas, especificar a largura mínima aceitável para os pulsos. Caso seja identificado um pulso muito estreito, isto é, com largura inferior à especificada (portanto,



incapaz de acionar uma porta), o programa deve reconhecê-lo e sinalizar ao projetista a ocorrência deste *spike*¹.

Estes simuladores devem detectar é a ocorrência de *rares*, que acontecem quando dois ou mais sinais geram um pulso indesejado na saída de uma porta, devido a problemas de atrasos diferentes ao longo de seus percursos. Ademais, o projetista pode verificar a ocorrência de *hazards*², que correspondem ao aparecimento de pulsos indevidos ocasionados por assincronismo nos sinais de acionamento das portas. Em fim, os projetistas devem poder, através do simulador, verificar circuitos seqüenciais assíncronos.

A estrutura básica dos simuladores lógicos é apresentada na figura 32. Eles compõem-se de quatro módulos fundamentais, quais sejam: o de **pré-processamento**, o **compilador de modelos**, o **controlador do simulador** e o de **pós-processamento**. O módulo de pré-processamento verifica a sintaxe da linguagem usada para descrever o circuito. O módulo compilador de modelos transforma a descrição de entrada numa representação interna compreensível ao simulador. O módulo de controle do simulador, é responsável pela análise do circuito simulado, através da avaliação das mudanças nos sinais do circuito ao longo do tempo. O módulo de pós-processamento formata os resultados para apresentação ao usuário de forma conveniente.

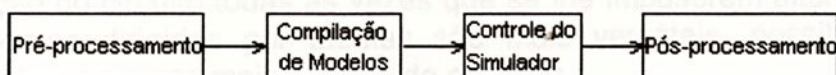


Figura 32 - Estrutura Básica de um Simulador em Nível de Portas

A análise de um circuito digital, a partir de um simulador lógico, inicia com a descrição deste circuito, através de uma linguagem especial ou de um sistema gráfico. Sobre esta descrição, o módulo de pré-processamento analisará o circuito do ponto de vista sintático. Este módulo não é capaz de identificar certos erros estruturais, nem erros funcionais. Dentre os erros estruturais não observados, está a ligação equivocada de portas, por exemplo, a ligação de uma porta P1 a uma porta P3, quando a porta P1 deveria ter sido ligada à porta P2. Erros funcionais, tais como a declaração de uma porta OU onde deveria-se declarar uma porta NE, também não podem ser descobertos pelo módulo de pré-processamento. Erros destes tipos só podem ser detectados após a execução do programa, através da verificação da incorreção das formas de onda obtidas. Por outro lado, erros estruturais, tais como a inexistência de conexão de sinal à entrada de determinada porta ou a saída de certa porta sem ligação à entrada de outra, devem ser reconhecidos e o usuário do simulador deve ser alertado para a ocorrência destes erros.

¹ - sinal com duração menor que o retardo de propagação e é produzido por um pulso na entrada da porta com duração menor que o retardo de propagação e maior que o retardo inercial.

² - sinal temporário e indesejado que pode ocorrer em circuitos lógicos devido a retardos diferentes e percursos diversos aos quais são submetidos os sinais do circuito.



Terminada a verificação sintática do circuito, gera-se a partir deste, um modelo interno para a simulação. Estes modelos podem estar sob a forma de código compilado ou tabelas.

Nos modelos de código compilado, a descrição do circuito é compilada numa sequência de instruções de computador, com estas instruções, o simulador é capaz de prever a resposta do circuito para um conjunto de entradas. Os simuladores de código compilado são verificadores funcionais, portanto, eles geram o resultado correto da simulação num único passo. Para seu correto funcionamento, estes simuladores necessitam que os componentes do circuito simulado estejam na sequência certa, isto é, se for alterado a sequência das portas, altera-se o resultado da simulação. Assim, é necessário um nivelamento do circuito de forma que as portas sejam organizadas para que a saída de cada porta seja avaliada e seu valor lógico atualizado antes que ele seja usado como entrada numa porta sucessora.

Nos modelo dirigido por tabelas, o circuito é convertido num conjunto de tabelas interligadas, as quais serão manipuladas pelo programa de simulação.

A principal vantagem de utilizar-se os simuladores de código compilado é a sua maior rapidez na solução de um circuito. Todavia, estes simuladores comungam uma série de desvantagens, quais sejam: a necessidade de nivelamento dos circuitos a serem simulados; a limitação da simulação de apenas circuitos combinatórios ou sequenciais síncronos; a necessidade de recompilação do circuito todas as vezes que se lhe impuserem alterações. Já, os simuladores dirigidos por tabelas são mais versáteis, possibilitando a simulação de uma gama mais variada de circuitos.

Os simuladores dirigidos por tabelas têm sua principal aplicação na verificação de projetos, enquanto os de código compilado são usados, principalmente, para modelar sistemas já existentes, por exemplo, aqueles cuja construção é relativamente cara e dos quais se faz necessário estudar o comportamento sob condições de falhas.

Após compilado o circuito, ou suas tabelas devidamente montadas, o módulo de controle do simulador começa a trabalhar. Este módulo devolverá sinais de saída em respostas a um dado conjunto de entradas, desde que duas funções básicas sejam executadas, as quais são:

- 1) avaliar os efeitos de alterações nos sinais do circuito sendo simulado;
- e
- 2) escalar eventos (mudanças nos sinais) numa ordem correta de tempos.

De maneira geral, diversas saídas de um circuito mudam de valor simultaneamente. Como a maioria dos computadores não processam eventos simultâneos, a maneira que se usa é, "congelar", momentaneamente, o tempo interno do simulador e atualizar os valores lógicos de um sinal por vez. Uma vez que se tenha atualizado todos os valores lógicos pertinentes, só então, os efeitos destas alterações podem ser avaliadas, para o instante de tempo sucessor.



c) *Simulação em Nível de Registradores (RT)* [19]

O nível RT se caracteriza pela representação de um sistema como um conjunto de módulos. Cada um destes módulos são construídos a partir de funções digitais, tais como, registradores, contadores, decodificadores, *multiplexadores*, elementos aritméticos e lógica de controle. Os vários módulos são interconectados a partir de caminhos de dados e de controle comuns, para formar um sistema computacional digital.

Para descrever um sistema digital em termos de funções tais como somadores, decodificadores e registradores, é necessário o emprego de uma notação matemática de alto nível. Nesta notação, os registradores são escolhidos para serem os componentes primitivos no sistema digital. Assim, é necessário apenas descrever o fluxo de informações e o processamento dos dados armazenados nos registradores. O método lógico de transferência entre registradores usa um conjunto de expressões e comandos que lembram os comandos utilizados nas linguagens de programação.

O termo registrador, não se refere apenas aos vistos na lógica seqüencial, mas também a registradores de deslocamento, contadores e unidades de memória. Um contador é considerado como um registrador cuja função é incrementar de uma unidade a informação armazenada nele. Uma unidade de memória é considerada como um conjunto de registradores onde pode ser armazenada informação. Um único *flip-flop*, também conhecido como registrador biestável, é entendido como um registrador de um *bit*. De fato, os *flip-flops* e todas as portas a eles associadas, de qualquer circuito seqüencial, são considerados registradores, neste método.

Linguagens de transferência entre registradores ou linguagens computacionais para a descrição de *hardware*, também são conhecidas como linguagens de descrição de *hardware* (HDLs). O desenvolvimento destas linguagens teve início no começo da década de 1970 e no decorrer dos anos 70 surgiram diversas HDLs. No final desta década, com o intuito de organizar a situação, o departamento de defesa norte americano (DoD) deu início a um programa para a normatização dos projetos a serem apresentados ao DoD. Deste programa surgiu a linguagem VHDL, que vem se tornando um padrão internacional para a descrição de sistemas digitais, desde o nível comportamental de suas arquiteturas até o nível estrutural de chaves bidirecionais.

A maioria das HDLs são definidas para permitirem tanto descrições estruturais como comportamentais. Tal flexibilidade tem a ver tanto com a distinção entre os blocos operacional e de controle, como com os objetivos da simulação no nível de registradores, os quais são: servir à validação e/ou avaliação de arquiteturas, para o que uma descrição comportamental, que abstraia detalhes do *hardware* real, é mais conveniente, e de organizações alternativas para uma dada arquitetura, para o que descrições estruturais são exigidas. No nível estrutural o sistema é descrito como uma interconexão de elementos reais de *hardware*: registradores, somadores, *multiplexadores*, etc. no bloco operacional, e PLAs, memórias, etc. no bloco de controle.

Comum a quase todas as definições deste nível é o aparecimento explícito de estruturas de controle, uma vez que um sistema digital consiste de um bloco operacional e de um bloco de controle. O bloco de controle



normalmente é descrito de forma comportamental. Cada linguagem, no entanto, define sua estrutura particular para expressar o controle.

Como já mencionado acima, o modelamento de sistemas digitais podem ser feito em nível estrutural ou comportamental. Modelos estruturais são aqueles nos quais os componentes do sistema são descritos, ou representados, pela interconexão de outros elementos (ou dispositivos) mais primitivos, previamente existentes.

Os modelos comportamentais requerem menos detalhamento sobre o *hardware* do que os modelos estruturais. Logo, utilizando-se um nível de abstração mais elevado, há possibilidade de analisar sistemas maiores, em menos tempo e em suas fases mais incipientes de projeto. Desta forma, os modelos comportamentais são os preferíveis para as análises no nível de sistemas.

d) Simulação Multi-Nível [12]

A simulação de sistemas muito complexos em níveis inferiores de projeto (lógico ou elétrico) tem no grande tempo de processamento sua maior desvantagem. Uma solução possível é a realização de uma simulação mais detalhada apenas nas partes mais críticas do sistema, utilizando-se um nível mais abstrato de projeto para o restante do sistema. Além disto, o projeto de tais sistemas exige metodologias que permitam descrições modulares e um processo de refinamentos sucessivos dos diversos módulos. Um refinamento pode tanto manter o mesmo nível de projeto, quando um módulo é dividido em vários submódulos, como passar a uma descrição num nível inferior de projeto, mais próximo da implementação final. Pelos motivos expostos, é útil para o processo de projeto, que um modelo do sistema, contendo módulos descritos nos diferentes níveis de projeto, possa ser simulado. Tal facilidade exige técnicas de simulação multi-nível, que passaram a atrair um maior interesse a partir do início desta década.

A literatura relata pelo menos três tipos básicos de simuladores multi-nível, que utilizam técnicas diversas para sua implementação:

- a) simuladores que incluem unicamente os níveis discretos tradicionais de simulação, isto é, do nível lógico para cima;
- b) simuladores que combinam descrições nos níveis discretos tradicionais com o nível de chaves;
- c) simuladores que combinam níveis discretos (lógico e/ou chaves) com o nível elétrico, que é contínuo.

5 - Síntese Automática

5.1 - Introdução

A síntese automática realiza transformações entre níveis de especificação de um sistema digital, chegando sistematicamente cada vez mais próximo de uma realização física para esse sistema, introduzindo a cada transformação mais detalhes estruturais ou geométricos e mantendo, ao



mesmo tempo, o comportamento desejado para o sistema. A síntese automática pode ser considerada como um método de validação de projeto, pois um sistema digital assim gerado tem garantida sua correção.

A síntese automática, com o passar do tempo, vem incorporando ferramentas que atuam em níveis cada vez mais elevados. Foi englobado inicialmente, o nível lógico tanto estrutural (interconexão de portas lógicas) como comportamental (sinal de saída como uma função booleana das entradas), visando alcançar os níveis mais altos de abstração.

Um sistema ideal deveria iniciar:

- a) o processo de síntese a partir de uma especificação comportamental do circuito em um alto nível de abstração;
- b) gerar sucessivamente diversas soluções em níveis cada vez mais baixos, escolhendo aquela solução que melhor se adaptasse às restrições de desempenho, área, velocidade e potência dissipada;
- c) produzir, em um intervalo de tempo significativamente curto, um circuito final tão bom ou até melhor que um projetado por especialistas.

Um sistema ideal como mencionado acima ainda não existe. Falta uma compreensão melhor do processo criativo de síntese para que possam ser desenvolvidas ferramentas com tal capacidade.

5.2 - Caracterização das Ferramentas de Síntese Automática

[19]

Síntese funcional

O objetivo da síntese funcional é transformar uma descrição comportamental do circuito, geralmente dada sob a forma de um algoritmo, em uma representação estrutural. A representação estrutural é, normalmente, apresentada no nível RT ou no de portas. Quando este objetivo não pode ser alcançado integralmente, o produto final é uma representação mista (estrutural e comportamental), comumente no nível RT.

A descrição comportamental do circuito a ser sintetizado tem forma semelhante a um programa de alto nível, escrito em uma linguagem de descrição de *hardware*, como o VHDL. Nestas linguagens, os recursos necessários para armazenar e transportar dados são representados como variáveis e os recursos para manipular dados aparecem como operações semelhantes a operações encontradas em linguagens de alto nível. O circuito gerado é representado com o auxílio de blocos estruturais, equações lógicas, tabelas e listas.

O procedimento mais comum nesse nível é dividir o circuito em um bloco operacional e um bloco de controle. O bloco operacional realiza o fluxo de dados descrito (variáveis, operadores e suas ligações) e o bloco de controle realiza o seqüenciamento entre as operações. Na aplicação desse procedimento é essencial a realização de otimizações. Otimizações são vitais para a eficiência do processo de síntese e constituem o principal interesse das pesquisas na área.



Síntese Lógica

Para que uma ferramenta seja classificada como de síntese lógica, o importante é que o produto final seja representado no nível lógico.

A síntese lógica é vista como:

- a) uma transformação de uma descrição comportamental do circuito, dada sob a forma de equações lógicas (funções booleanas), em uma representação estrutural no nível de portas lógicas;
- b) uma transformação de uma descrição estrutural do circuito, ao nível de transferência entre registradores, em uma representação no nível de portas lógicas;
- c) uma transformação de uma descrição do circuito, sob a forma de funções booleanas ou portas lógicas, em uma representação otimizada do circuito no mesmo nível.

Assim como na síntese funcional, a simples alocação de recursos pode ser trivial. Problemas aparecem na otimização, também conhecida como minimização lógica, cuja solução é reconhecidamente NP-completa, ou seja, inviável de ser realizada por exaustão (determinação de todas as possibilidades e escolha da mais simples).

Para contornar a complexidade do problema da minimização lógica, dois caminhos têm sido seguidos: aplicação de otimizações globais ou de otimizações locais.

Compiladores de Silício

O grande sonho daqueles que trabalham com o desenvolvimento de ferramentas de síntese automática é a construção de um compilador de silício, nome atribuído a um instrumento que permitiria a obtenção automática das máscaras de um circuito a partir de sua descrição numa linguagem de alto nível, ou seja, um sistema que parta de uma descrição lógica (estrutural ou comportamental) ou até mesmo geométrica, de um alto nível de abstração, e gere uma descrição geométrica no nível de *layout*.

Comercialmente, os produtos oferecidos como compiladores de silício podem ser caracterizados como sistemas baseados numa biblioteca de células onde os componentes elementares para a construção do circuito são células parametrizadas.

Síntese de layout

A síntese de *layout* é o processo através do qual transforma-se uma descrição estrutural de um circuito, em sua correspondente descrição geométrica especialmente no nível de máscaras.

É nessa área onde se encontra o maior número de ferramentas comerciais disponíveis. A razão para o sucesso e o grande número de produtos em comparação às sínteses lógica e funcional se deve aos seguintes fatores:

- o processo de síntese no eixo geométrico, principalmente nos níveis de abstração mais próximos da realização física, se encontra



dominado e equacionado, sendo desta forma relativamente fácil estabelecer procedimentos e algoritmos para a sua realização.

- sem ferramentas para tratamento da geometria do circuito nos níveis menos abstratos, a integração de circuitos VLSI seria inviável. O mercado, altamente receptivo, facilitou e até acelerou o desenvolvimento de geradores de *layout*.

6 - Teste [19]

O teste é um dos métodos de validação do sistema digital, através do qual é realizado um confronto entre a especificação ou descrição do sistema e a implementação obtida. Este confronto é executado, de forma geral, pela aplicação de excitações ao circuito e pela análise das respostas obtidas, ou sua comparação com valores esperados.

As maiores dificuldades relacionadas ao teste de sistemas referem-se à sua realizabilidade e ao tempo envolvido na preparação e na aplicação dos testes, propriamente dita. É necessário assegurar: a *geração de testes* em um tempo viável, a *cobertura de falhas* em níveis aceitáveis, a *testabilidade* de todos os blocos funcionais internos ao circuito, e a *viabilidade econômica em termos de tempo* necessário à realização de testes. Estas condições podem ser atingidas pelo uso de técnicas adequadas de geração de testes e de projeto de circuitos integrados.

Ao longo dos processos de concepção e fabricação, um circuito integrado é submetido a procedimentos de teste para verificar seu funcionamento correto. Vários tipos de teste são efetuados nas seguintes etapas: *logo após a concepção*, na saída dos primeiros protótipos, tendo-se por objetivo revelar e diagnosticar as falhas de concepção ou, não existindo falhas, demonstrar a funcionalidade correta do circuito; em seguida, *durante a produção*, é necessário efetuar um teste de fim de fabricação com o objetivo de eliminar os circuitos defeituosos; após a *montagem dos circuitos em uma placa*, é necessário verificar que esta placa é funcionalmente correta; e assim por diante, até obter-se o sistema completo.

As técnicas empregadas para efetuar testes nos circuitos e sistemas durante as diversas fases citadas acima não são idênticas entre si. Assim como, a complexidade do teste elaborado e o tempo necessário à sua aplicação, são dependentes dos objetivos do mesmo. Estes objetivos podem ser a simples verificação da correção funcional do sistema; ou, detectando-se a existência de uma incorreção, o teste pode visar a identificação do componente ou do ponto interno defeituoso.

A geração de sinais programáveis nas entradas do circuito, que causam o aparecimento de um valor incorreto nos pontos de saída observáveis, se a falha estiver presente, podem ser obtidos das seguintes maneiras:

- testes exaustivos - aplicar todas as possíveis combinações às portas de entrada do circuito
- teste algorítmico - utilizar combinações escolhidas a partir de modelos de falhas e simulações executadas.
- teste aleatório - utilizar qualquer conjunto de combinações.



O Projeto das Chaves

Neste capítulo, é apresentado o projeto das chaves propostas para esta dissertação de mestrado.

No item 1, expõe-se a estrutura de controle da chave, e o protocolo de comunicação proposto.

No item 2, é exposto o projeto da chave em nível de transferência de registradores.

No item 3, o projeto é apresentado em nível lógico, ou seja, demonstra-se como são realizadas as operações lógicas dentro da chave. Ademais, expõe-se, o mecanismo responsável pelo sincronismo na troca de informações entre os processadores.

1 - Especificação do Projeto

Como já mencionado no capítulo 2, o projeto das chaves programáveis para a interconexão de processadores, se baseou na malha de processadores e chaves propostas pela metodologia ABACUS. Embora estas chaves estejam fortemente vinculadas ao ABACUS, não significa que a aplicação destas chaves se restrinja a esta metodologia, pelo contrário, elas podem ser aplicadas em diversos projetos que necessitem realizar a comunicação entre processadores.

Portanto, a tarefa das chaves é a de conectar vários processadores de um arranjo, de forma que eles possam trocar informações durante todo o tempo de processamento do arranjo.

Para facilitar a descrição deste tipo de chave, vamos abstrair da malha de chaves e processadores, apenas um subconjunto formado por quatro processadores (N, S, L e O) e uma chave.

Para que se possa estabelecer um duto de comunicação entre quaisquer dois dos quatro processadores (N, S, L e O) deve-se ter uma palavra de controle de 4 *bits*. Esta palavra de controle é apresentada na figura 33.

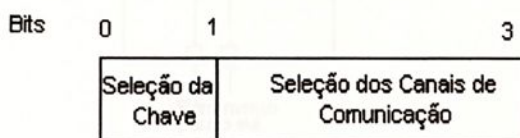


Figura 33 - Palavra de Controle

Pelo exposto na figura 33, vê-se que a palavra de controle fornece à chave 2 tipos de informação: a habilitação da chave e a dos canais de comunicação.

É importante ressaltar, que o número de *bits* necessários à seleção da chave, está intimamente ligada à quantidade de chaves utilizadas, ao contrário do total de *bits* responsáveis pela seleção do canal de comunicação, que é sempre fixo. Portanto, o tamanho da palavra de controle pode variar dependendo do total de chaves usadas.

Neste trabalho, utilizou-se apenas uma chave, logo faz-se necessário apenas 1 *bit* de seleção.

Portanto, o primeiro *bit* da palavra de controle responde pela habilitação ou não da chave e os demais pelos canais de comunicação.

Exigem-se 3 *bits* para a seleção dos dutos de transmissão de dados, pois à chave são ligados 4 canais e o objetivo é conectar 2 deles, portanto 6 possíveis combinações, logo 3 *bits* são suficientes.

Quando a palavra de controle chega na chave, analisa-se o sinal do primeiro *bit* e, de acordo com este sinal, o restante dos *bits* da palavra serão aceitos ou não.

Na chave deve existir um componente responsável pelo armazenamento dos 3 *bits* finais da palavra de controle. Esta **memória** deve ter capacidade de armazenamento suficiente para guardar estes *bits* durante todo o processo de comunicação entre os processadores.

Para que o processo de conexão dos processadores seja concretizado, é necessário que os 3 *bits* armazenados na memória passem por um processo de decodificação e transformem-se em quatro. Após serem decodificados, estes quatro *bits* servirão de entrada para o dispositivo responsável pela conexão dos canais ligados aos processadores.

Em linhas gerais, o esquema de uma chave quadrada, que liga processadores vizinhos numa mesma camada pode ser vista na figura 34.

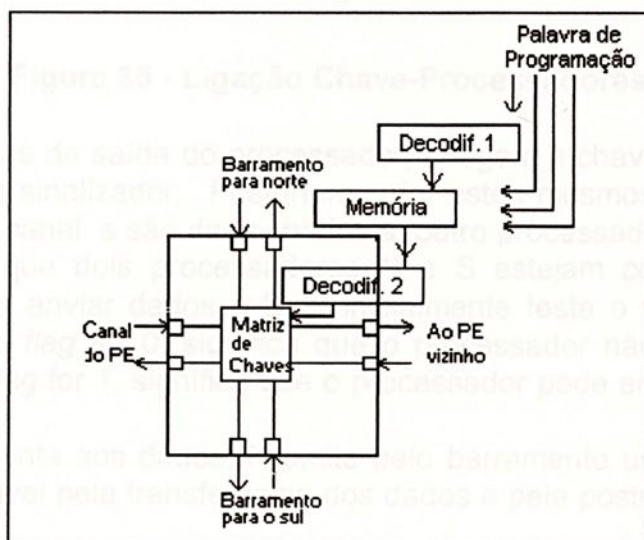


Figura 34 - Chave Quadrada

Realizada a especificação das chaves, passa-se à especificação do protocolo de comunicação dos processadores. Este protocolo tem por objetivo



sincronizar a troca de informações entre os processadores, ou seja, ordenar o tráfego das informações pela chave.

Cada um dos processadores do arranjo deve ter uma lógica interna, que permita:

- transmitir dados pelo barramento;
- gerador de pulsos;
- sinalizar seu *status*; e
- receber dados do barramento.

Necessita-se que cada um dos 4 canais que ligam os processadores N, S, L e O à chave possuam 6 linhas para a transmissão de dados. Destas 6 linhas, 3 serão utilizadas como entrada de dados para o processador e as outras 3 como saída. Portanto, chegam à chave um total de 24 linhas de dados. Na figura 35, tem-se um esquema de comunicação de um processador do arranjo.

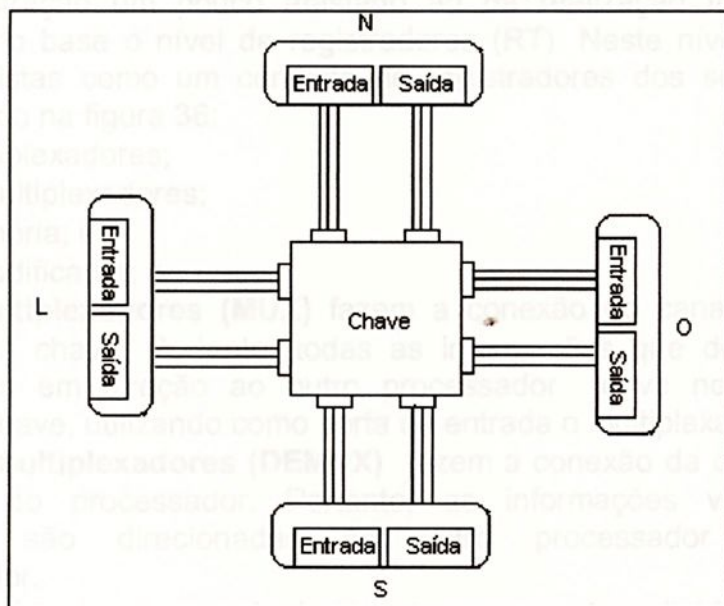


Figura 35 - Ligação Chave-Processadores

Pelas linhas de saída do processador, chegam à chave, dados, pulsos de *clock* e o *flag* sinalizador. Posteriormente, estes mesmos dados saem da chave por outro canal e são direcionados a outro processador.

Suponha que dois processadores N e S estejam conectados, então quando N deseja enviar dados a S, N inicialmente testa o valor de seu *flag* sinalizador. Se o *flag* for 0, significa que o processador não pode enviar os dados. Se este *flag* for 1, significa que o processador pode enviar dados pelos barramento.

Paralelamente aos dados, N emite pelo barramento uma seqüência de pulsos, responsável pela transferência dos dados e pela posterior recepção no processador S.

Depois de receber dos dados e pulsos, S altera o valor do *flag*, e o envia a N, ou seja, S sinaliza a N a recepção dos dados e o impede de enviar novas informações. O processador S, só irá alterar novamente o valor *flag* quando os dados forem colocados no seu interior (*buffer*).



É importante ressaltar que a troca de dados entre processadores pode se dar em ambos os sentidos, ou seja, tanto N como S recebem e enviam dados pelo barramento.

2 - O Projeto em Nível de Registradores

Como já foi dito, as chaves propostas neste trabalho são componentes que definem quais os processadores que deverão se comunicar durante o processamento de um arranjo. Assim, a função básica da chave, é a de criar um canal de comunicação entre quaisquer dois processadores.

Nosso intuito, aqui é o de apresentar a composição das chaves, em nível de abstração um pouco afastado ao da realização física, por isso tomamos como base o nível de registradores (RT). Neste nível, tais chaves podem ser vistas como um conjunto de registradores dos seguintes tipos, como esboçado na figura 36:

- multiplexadores;
- demultiplexadores;
- memória; e
- decodificador;

Os **multiplexadores (MUX)** fazem a conexão do canal de saída do processador à chave. Portanto, todas as informações que deixam um dos processadores em direção ao outro processador deve necessariamente passar pela chave, utilizando como porta de entrada o multiplexador.

Os **demultiplexadores (DEMUX)** fazem a conexão da chave ao canal de entrada do processador. Portanto, as informações vindas de um processador são direcionadas ao outro processador através do demultiplexador.

A **memória** é a responsável pelo armazenamento e distribuição de uma parte dos *bits* da palavra de controle ao decodificador2. Estes *bits* têm a função de selecionar os processadores a serem conectados pela chave.

O **decodificador1** é o componente da chave que recebe a outra parte da palavra de controle. Este componente tem a função de decodificar os *bits* que habilitam ou não a chave para o seu funcionamento.

O **decodificador2** tem a função de transformar as saídas da memória, para um formato compreensível ao multiplexador e demultiplexador, ou seja, este componente configura os multiplexadores e demultiplexadores a habilitarem os canais de transferência de dados.

Inicialmente chega na chave, mais precisamente ao decodificador1, uma parte palavra de controle. Esta parte referente ao endereço da chave, é decodificada e avaliada como verdadeiro se o endereço da chave coincide com o da palavra de controle.

Se o resultado da avaliação do decodificador for verdadeiro, ele habilita a memória a armazenar a outra parte da palavra de controle. Este pedaço da palavra (controle de troca) contém informação que definirá quais os processadores irão trocar dados.



O controle de troca é então enviado ao decodificador2, que os converterá nos seletores dos multiplexadores e demultiplexadores. Com seus seletores programados, os multiplexadores e demultiplexadores conectam os respectivos processadores, habilitando-os para a troca de informação.

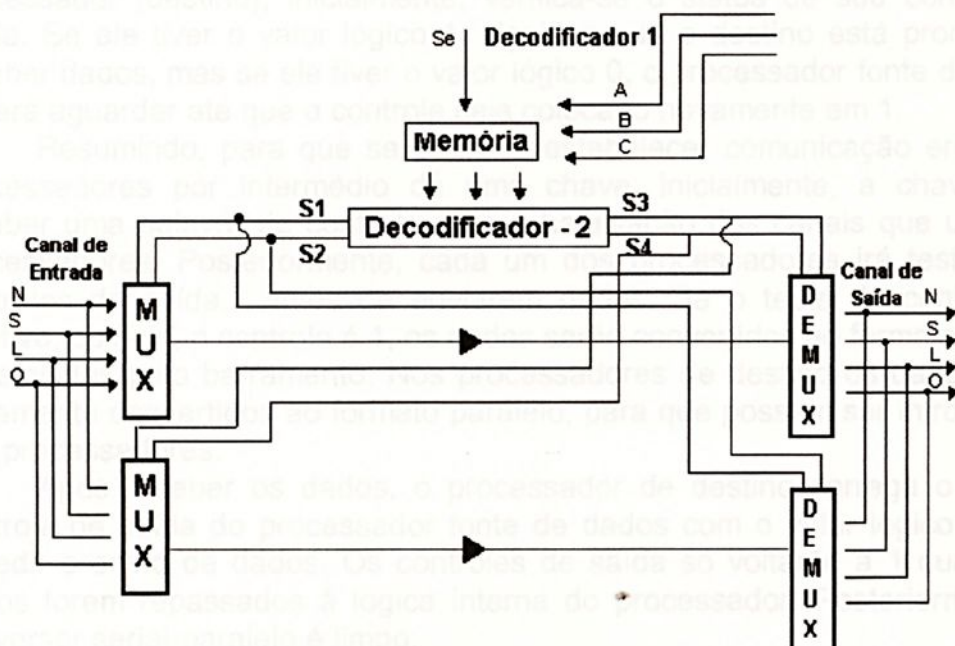


Figura 36 - Chave em Nível RT

Juntamente como o projeto das chaves, faz-se necessário descrever a composição do protocolo de comunicação dos processadores do arranjo. Este protocolo tem a função de receber e enviar dados a outro processador.

Para que esta troca de informações seja possível, os processadores devem possuir:

- um registrador paralelo-serial;
- um registrador serial-paralelo;
- um relógio (pulso);
- controle de entrada de dados; e
- controle de saída de dados.

O **conversor paralelo-serial** transforma os dados paralelos do *buffer* de saída do processador, para a forma serial, para que as mesmas possam trafegar pelo barramento, até o processador de destino.

O **conversor serial-paralelo** faz exatamente o oposto, ele transforma as informações seriais vindas do barramento, para a forma paralela, para que elas possam ser armazenadas no *buffer* de entrada do processador de destino.

O **relógio (pulso)** é contador síncrono, responsável pela geração de pulsos aos conversores serial-paralelo e paralelo-serial, a fim de que estes registradores possam receber ou enviar os dados pelo barramento.

O **controle de entrada** é um *flag* que indica ao processador local a presença de uma palavra no conversor serial-paralelo. Além de sinalizar a



presença de dados, o controle de entrada reinicializa o conversor serial-paralelo, habilitando-o a receber novos dados.

Já o **controle de saída** é um *flag* sinalizador do *status* do processador remoto. Este *flag* tem a função de indicar ao processador local se é ou não possível a transferência de dados pelo barramento.

Portanto, quando um processador deseja enviar dados a um outro processador (destino), inicialmente, verifica-se o *status* de seu controle de saída. Se ele tiver o valor lógico 1, significa que o destino está pronto para receber dados, mas se ele tiver o valor lógico 0, o processador fonte de dados deverá aguardar até que o controle seja colocado novamente em 1.

Resumindo, para que se possa estabelecer comunicação entre dois processadores por intermédio de uma chave, inicialmente, a chave deve receber uma palavra de controle para a habilitação dos canais que unem os processadores. Posteriormente, cada um dos processadores irá testar seus controles de saída, antes de enviarem dados. Se o teste do controle for positivo, ou seja, o controle é 1, os dados serão convertidos ao formato serial e transmitidos pelo barramento. Nos processadores de destino os dados serão novamente convertidos ao formato paralelo, para que possam ser introduzidos nos processadores.

Após receber os dados, o processador de destino carrega o *flag* do controle de saída do processador fonte de dados com o valor lógico 0, para impedir o envio de dados. Os controles de saída só voltarão a 1 quando os dados forem repassados à lógica interna do processador. Posteriormente, o conversor serial-paralelo é limpo.

Na figura 37, é apresentado o protocolo de comunicação.

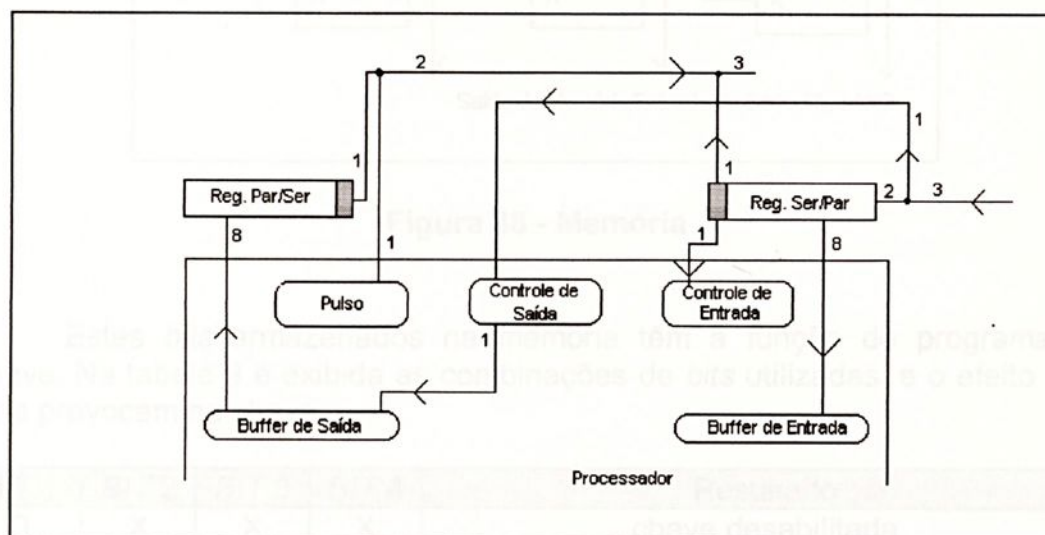


Figura 37 - Protocolo de Comunicação

4 - O Projeto em Nível Lógico

4.1 - Chaves

Nosso intuito aqui, é o de apresentar a composição das chaves em nível mais próximo ao da realização física. O nível lógico é, na realidade, o objetivo final desta dissertação.

Mais uma vez, vamos considerar um arranjo de 4 processadores (N, S, L e O) e uma chave. Portanto, é necessário uma palavra de controle de 4 bits.

Quando a palavra de controle chega na chave, o primeiro bit desta palavra é colocado na entrada de habilitação da memória e os demais são ligados às entradas de dados da memória. Cada um dos 3 flip-flops da memória são responsáveis pelo armazenamento de um dos bits da palavra de controle. A figura 38, mostra esta ligação.

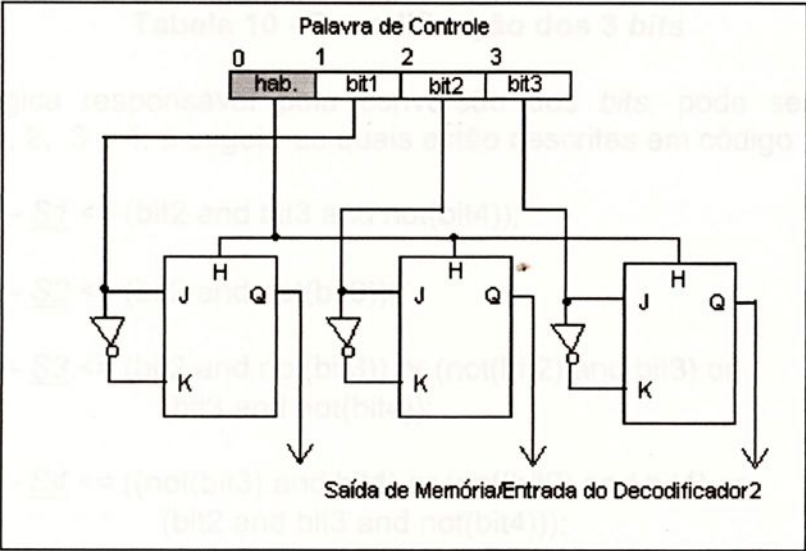


Figura 38 - Memória

Estes bits armazenados na memória têm a função de programar a chave. Na tabela 9 é exibida as combinações de bits utilizadas, e o efeito que elas provocam na chave.

BIT 1	BIT 2	BIT 3	BIT 4	Resultado
0	X	X	X	chave desabilitada
1	0	0	1	N ligado a S e vice-versa
1	0	1	0	N ligado a L e vice-versa
1	0	1	1	N ligado a O e vice-versa
1	1	0	0	S ligado a L e vice-versa
1	1	0	1	S ligado a O e vice-versa
1	1	1	0	L ligado a O e vice-versa

Tabela 9 - Bits da Palavra de Controle

Após serem armazenados na memória, os sinais *bit2*, *bit3* e *bit4* (3 *bits* da palavra de controle) servirão como entradas para um decodificador. Este decodificador irá transformar estes 3 *bits* em 4, para que estes novos sinais possam interligar os devidos canais. A tabela 10, mostra o resultado desta transformação

BIT 2	BIT 3	BIT 4	S1	S2	S3	S4	CONEXÃO
0	0	1	0	0	0	1	N ligado a S e vice-versa
0	1	0	0	0	1	0	N ligado a L e vice-versa
0	1	1	0	0	1	1	N ligado a O e vice-versa
1	0	0	0	1	1	0	S ligado a L e vice-versa
1	0	1	0	1	1	1	S ligado a O e vice-versa
1	0	0	1	0	1	1	L ligado a O e vice-versa

Tabela 10 - Decodificação dos 3 *bits*

A lógica responsável pela conversão dos *bits*, pode ser vista nas equações 1, 2, 3 e 4, a seguir, as quais estão descritas em código VHDL.

Equação 1 - $S1 \leq (\text{bit2 and bit3 and not}(\text{bit4}));$

Equação 2 - $S2 \leq (\text{bit2 and not}(\text{bit3}));$

Equação 3 - $S3 \leq (\text{bit2 and not}(\text{bit3})) \text{ or } (\text{not}(\text{bit2}) \text{ and bit3}) \text{ or } (\text{bit3 and not}(\text{bit4}));$

Equação 4 - $S4 \leq ((\text{not}(\text{bit3}) \text{ and bit4}) \text{ or } (\text{not}(\text{bit2}) \text{ and bit4}) \text{ or } (\text{bit2 and bit3 and not}(\text{bit4})));$

Nestas equações, aparecem as variáveis **bit2**, **bit3**, e **bit4** representando as saídas da memória e **S1**, **S2**, **S3** e **S4** representando os *bits* transformados.

A saída do decodificador, ou seja, S1, S2, S3 e S4, é ligada aos seletores de 2 multiplexadores e de 2 demultiplexadores.

S1 e S2, atuam como seletores para um multiplexador e um demultiplexador. S3 e S4, como seletores dos outros multiplexador e demultiplexador.

Após serem habilitados, os seletores S1, S2, S3 e S4 induzem os multiplexadores a receberem informações de apenas dois dos quatro canais ligados à chave, por exemplo, do norte e do sul. Posteriormente, estes mesmos seletores induzem os demultiplexadores a desviarem as informações de entrada para dois dos quatro canais de dados, neste caso norte e sul, sendo que o canal norte recebe a informação vinda do Sul, e o Sul recebe a informação vinda do Norte.

Portanto, os dados vindos de um processador entram por um dos canais de entrada do multiplexador e são direcionados ao demultiplexador, que se encarregará de colocá-los em um dos canais de saída.



4.2 - Protocolo

Como já discutido, para que a troca de informações seja possível, requer-se que os processadores possuam:

- um registrador paralelo-serial de 9 *bits*;
- um registrador serial-paralelo de 9 *bits*;
- um relógio (pulso);
- controle de entrada;
- controle de saída de dados; e
- seis linhas de dados, três para a entrada e três para a saída.

Para realizar a descrição do protocolo em nível lógico, se utilizará um conjunto formado por 4 processadores N, S, L e O e uma chave.

Para que um processador, por exemplo N, possa transmitir uma palavra de 8 *bits* a outro processador, S por exemplo, requer-se que existam dados no *buffer* de saída de N, que o controle de saída de N esteja em nível 1 e, principalmente, que a chave esteja habilitada.

O controle de saída é um componente de N, que recebe o sinal do primeiro *bit* de dados que chega no registrador serial/paralelo de S. Este *bit* é transmitido a N através de uma das 3 linhas de entrada de dados do processador.

Portanto, existindo dados nos *buffers* de N e com o seu controle de saída habilitado, inicia-se o processo de transmissão dos dados.

Inicialmente, o registrador paralelo/serial de N é limpo, ou seja, sua saída é levada a 0. Em seguida, os dados de N são colocados no registrador por meio de um sinal de *preset* sobre cada um dos *flip-flops* que compõem o registrador paralelo/serial (figura 39).

É importante observar que a palavra de dados a ser transmitida é de 8 *bits* e que o registrador paralelo/serial armazena 9 *bits*. Este *bit* adicional do registrador, no caso o primeiro dos nove, é responsável pelo sincronismo do sistema.

Assim, sempre que uma palavra de 8 *bits* é transmitida serialmente, inicialmente transfere-se o *bit* de controle, em nível 1, em seguida os 8 *bits* de dados.

Paralelamente à carga dos dados, o registrador paralelo/serial recebe de N uma seqüência de pulsos que alimenta o registrador a enviar a palavra de dados pelo barramento.

Figura 40 - Registrador Serial/Paralelo



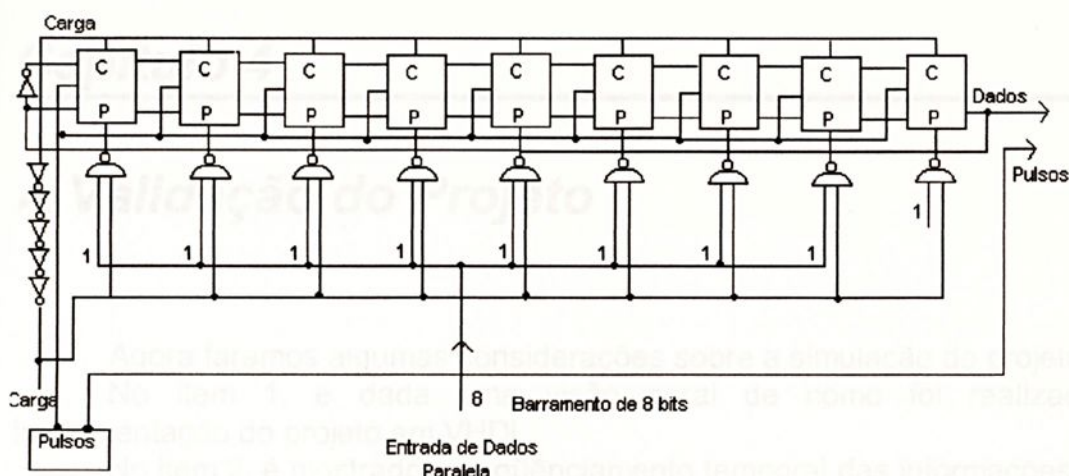


Figura 39 - Registrador Paralelo/Serial

Assim, por uma linha do barramento irão trafegar os 9 *bits* da palavra de dados. Por outra linha, irão trafegar os pulsos que excitam o registrador paralelo/serial. Estas duas linhas, que partem do processador N, chegarão através da chave no registrador serial/paralelo de S. A sequência de pulsos faz com que o registrador serial/paralelo armazene gradativamente cada um dos 9 *bits* transmitidos.

Quando o primeiro *bit*, no caso o de controle, chegar no último *flip-flop* do registrador serial/paralelo (figura 40) a lógica interna do processador S irá transmitir, pelo barramento, um novo sinal ao controle de saída do processador N, informando-o que os dados chegaram em S, e obviamente provocando término da transmissão dos dados.

Paralelamente à emissão do sinal de controle, a lógica interna de S faz com que os dados armazenados no registrador serial/paralelo sejam transferidos ao *buffer* de entrada de dados. Esta mesma lógica que recebe os dados se encarrega de limpar o registrador local e enviar outro sinal ao controle de saída do processador origem de dados, habilitando o envio de novos dados.

Vale observar que ambos os processadores podem enviar e receber dados pelo barramento ao mesmo tempo.

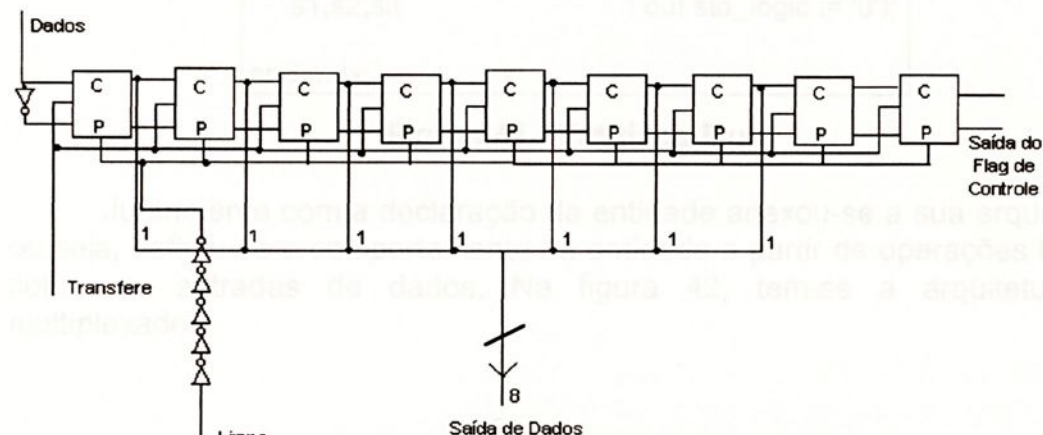


Figura 40 - Registrador Serial/Paralelo



Capítulo 4

A Validação do Projeto

Agora faremos algumas considerações sobre a simulação do projeto.

No item 1, é dada uma visão geral de como foi realizada a implementação do projeto em VHDL.

No item 2, é mostrado o seqüenciamento temporal das informações pelo sistema.

1 - A Descrição do Projeto em VHDL

O projeto proposto nesta dissertação, as chaves e o protocolo de comunicação foram totalmente implementados na linguagem de descrição de *hardware* VHDL.

O projeto das chaves foi modelado em VHDL estrutural, ou seja, cada um dos módulos (memória, decodificador, multiplexador e demultiplexador) foi descrito utilizando-se de operações lógicas sobre as entradas de dados.

Cada um destes módulos foi projetado separadamente e representados no programa através de entidades. Na figura 41 tem-se a entidade que representa o multiplexador.

```
entity mux is
port(a,b           : in std_logic := '0';
     ent11, ent12, ent13 : in std_logic := '0';
     ent21, ent22, ent23 : in std_logic := '0';
     ent31, ent32, ent33 : in std_logic := '0';
     ent41, ent42, ent43 : in std_logic := '0';
     s1,s2,s3         : out std_logic := '0');
end mux;
```

Figura 41 - Entidade Mux

Juntamente com a declaração da entidade anexou-se a sua arquitetura, ou seja, definiu-se o comportamento da entidade a partir de operações lógicas sobre as entradas de dados. Na figura 42, tem-se a arquitetura do multiplexador.


```

architecture mux of mux is
begin
  process(ent11,ent12,ent13,ent21,ent22,ent23,ent31,ent32,ent33,ent41,ent42,ent43)
  variable and1,and2,and3,and4 : std_logic := '0';
  begin
    and1 := not(a) and not(b);
    and2 := not(a) and b;
    and3 := a and not(b);
    and4 := a and b;
    s1 <= (and1 and ent11) or (and2 and ent21) or (and3 and ent31) or (and4 and
ent41) after 6 ns;
    s2 <= (and1 and ent12) or (and2 and ent22) or (and3 and ent32) or (and4 and
ent42) after 6 ns;
    s3 <= (and1 and ent13) or (and2 and ent23) or (and3 and ent33) or (and4 and
ent43) after 6 ns;
  end process;
end mux;

```

Figura 42 - Arquitetura do Multiplexador

Definidas todas as entidades e arquiteturas que representam cada um dos módulos da chave, criou-se outra entidade chamada **chave**, que une cada um dos módulos projetados. A entidade chave possui as seguintes entradas:

- 4 *bits* da palavra de controle.
- 12 canais de entrada de dados
- 12 canais responsáveis pela saída de dados.

Na figura 43, tem-se esta entidade.

```

entity chave is
port(memoria1,memoria2,memoria3 : in std_logic := '0';
  enable          : in std_logic := '0';
  e11,e12,e13     : in std_logic := '0';
  e21,e22,e23     : in std_logic := '0';
  e31,e32,e33     : in std_logic := '0';
  e41,e42,e43     : in std_logic := '0';
  s11,s12,s13     : out std_logic;
  s21,s22,s23     : out std_logic;
  s31,s32,s33     : out std_logic;
  s41,s42,s43     : out std_logic);
end chave;

```

Figura 43 - Entidade representando a Chave

Já o projeto do protocolo de comunicação dos processadores, foi descrito parte estrutural e parte comportamentalmente.

A lógica interna do processador, responsável pela entrada de dados no registrador paralelo-serial, verificação do controle de saída, geração de pulsos, sinalização da chegada de dados, etc. foi projetada em nível



comportamental sobre a entidade processador. Na figura 45, é apresentada esta entidade.

```
entity processador is
port(inf_part      : in dados := "00000000"; -- Informação De Saida Do Proc.
     inf_cheg      : in dados1 ;              -- Inf De Entrada Do Proc.
     registrador1   : inout dados;             -- DADOS DE Entrada P/ O RPS
     registrador2   : out dados := "00000000"; -- Dados De Saida Do Rsp
     hab_rps        : out std_logic := '1';    -- Carga Do Rps
     pulso          : out std_logic := '0';    -- Relogio
     rsp_rem        : in std_logic := '1';    -- Flag Do Rsp Remoto
     rsp_local      : inout std_logic := '0'); -- Flag Do Rsp Local
end processador;
```

Figura 43 - Entidade Processador

Já os registradores serial-paralelo e paralelo-serial foram descritos utilizando-se operações lógicas sobre as entradas. As entidades que descrevem estes registradores são a RSP e RPS respectivamente.

Finalmente, utilizou-se uma entidade chamada **total** que integra as entidades chave, processador, rps e rsp. Esta entidade emula um sistema composto por 4 processadores, N, S, L e O e uma chave. Esta entidade é mostrada na figura 44.

```
entity total is
port(busNE1,busNE2,busNE3      : inout std_logic := '0';
     busNS1,busNS2,busNS3      : inout std_logic := '0';
     busSE1,busSE2,busSE3      : inout std_logic := '0';
     busSS1,busSS2,busSS3      : inout std_logic := '0';
     busLE1,busLE2,busLE3      : inout std_logic := '0';
     busLS1,busLS2,busLS3      : inout std_logic := '0';
     busOE1,busOE2,busOE3      : inout std_logic := '0';
     busOS1,busOS2,busOS3      : inout std_logic := '0';
     cont2,cont3,cont4          : in std_logic := '0';
     cont1                      : in std_logic := '0';
     wordN                      : in dados := "00000000";
     wordS                      : in dados := "00000000";
     wordL                      : in dados := "00000000";
     wordO                      : in dados := "00000000";
     wordN_cheg                 : out dados := "00000000"
     wordS_cheg                 : out dados := "00000000"
     wordL_cheg                 : out dados := "00000000";
     wordO_cheg                 : out dados := "00000000");
end total;
```

Figura 44 - Entidade Total

Basicamente, a entidade total funciona como um sistema, onde inicialmente é entrado uma palavra de controle de 4 *bits*, que programa a chave a habilitar dois dos quatro canais ligados aos processadores. A seguir, é



colocada uma palavra de 8 *bits* no *buffer* de entrada dos processadores conectados. Após um período de tempo estes 8 *bits* aparecem nos *buffers* de saída dos processadores receptores dos dados.

Abaixo, tem-se o código VHDL completo para o sistema.

PACKAGE ÚTIL

```
use work.std_logic_1164.all;
package util is
    type dados is array(0 to 7) of std_logic; -- 8 bits de dados
    type dados1 is array(0 to 8) of std_logic; -- 9 bits -> dados + Flag
end util;
```

PROTOCOLO DE COMUNICAÇÃO DO PROCESSADOR

```
use work.util.all;
use work.std_logic_1164.all;
```

***** ENTIDADE DO PROTOCOLO *****

```
entity processador is
    port(inf_part      : in dados := "00000000"; -- Informação De Saída Do Proc.
          inf_cheg     : in dados1;              -- Inf De Entrada Do Proc.
          registrador1 : inout dados;             -- DADOS DE Entrada P/ O RPS
          registrador2 : out dados := "00000000"; -- Dados De Saída Do Rsp
          hab_rps      : out std_logic := '1';    -- Carga Do Rps
          pulso        : out std_logic := '0';    -- Relógio
          rsp_rem      : in std_logic := '1';    -- Flag Do Rsp Remoto
          rsp_local    : inout std_logic := '0'); -- Flag Do Rsp Local
end processador;
```

***** ARQUITETURA DA INTERFACE *****

```
architecture processador of processador is
    signal ativa_relogio : std_logic := '0';
    signal aux1          : std_logic := '1';
    signal aux_pulso     : std_logic := '0';
    signal habilita      : std_logic := '0';
begin
    process (aux1,ativa_relogio,aux_pulso,inf_cheg,rsp_rem)
```


Aux1 - controle do sinal de carga

ativa_relogio - controla o assionamento do relógio

aux_pulso - controle dos pulsos do relógio

inf_cheg - armazena as informacoes oriundas do rsp

rsp_rem - sincroniza o envio e recebimento de informacoes

```
variable aux : std_logic := '1';  
variable contador : std_logic := '0';  
variable armazena : dados;  
begin
```

MANIPULACAO DOS DADOS DE ENTRADA DO PROCESSADOR

```
if (rsp_rem = '0') then  
  if (aux = '1') then  
    aux := not(aux);  
    hab_rps <= aux after 1 ns; -- habilita carga  
    aux1 <= aux after 1 ns;  
    registrador1 <= inf_part; -- coloca dados no registrado PS  
  elsif aux = '0' then  
    aux := not(aux);  
    ativa_relogio <= aux after 10 ns; -- ativa os pulso de clock  
    aux1 <= aux after 6 ns;  
    hab_rps <= aux after 6 ns; -- desabilita carga  
    aux := 'X';  
  end if;  
  if (ativa_relogio = '1') then  
    if contador = '0' then  
      contador := not(contador);  
      aux_pulso <= contador after 2 ns;  
      pulso <= contador after 2 ns; -- pulsos do relógio  
    else  
      contador := not(contador);  
      aux_pulso <= contador after 5 ns;  
      pulso <= contador after 5 ns; -- pulsos do relógio  
    end if;  
  end if;  
end if;
```

MANIPULACAO DOS DADOS QUE CHEGAM NO PROCESSADOR

```
if (rsp_rem = '1') then  
  aux := '1'; -- reinicializacao para receber  
  aux1 <= '1' and '1'; -- novas informacoes  
  aux_pulso <= '0' and '1';  
  pulso <= '0' and '1';  
  ativa_relogio <= '0' and '1';  
  contador := '0';  
end if;
```



```

if inf_cheg(8) = '1' then
    registrador2(0) <= inf_cheg(0);
    registrador2(1) <= inf_cheg(1); -- informacoes do rsp sao
    registrador2(2) <= inf_cheg(2); -- repassadas para o processador
    registrador2(3) <= inf_cheg(3);
    registrador2(4) <= inf_cheg(4);
    registrador2(5) <= inf_cheg(5);
    registrador2(6) <= inf_cheg(6);
    registrador2(7) <= inf_cheg(7);
    rsp_local <= inf_cheg(8) after 5 ns; -- habilita um clear no rsp
    habilita <= '1' and '1' after 7 ns; -- habilita a recepcao de novos dados
end if;
if habilita = '1' then
    rsp_local <= '1' and '0' after 5 ns;
    habilita <= '1' and '0';
end if;

end process;
end processador;

```

FLIP-FLOP D

```

use work.util.all;
use work.std_logic_1164.all;

```

***** ENTIDADE FLIP-FLOP D *****

```

entity flip_flop_d is
port(j
    preset      : in std_logic := '0';
    clear       : in std_logic := '1';
    clock       : in std_logic := '0';
    q           : inout std_logic := '0';
    qb          : inout std_logic := '1');
end flip_flop_d;

```

***** ARQUITETURA DO FLIP-FLOP D *****

```

architecture flip_flop_d of flip_flop_d is
signal nand1,nand2 : std_logic := '1';

```

```

begin
    process (clock,preset,clear,q,qb,nand1,nand2)
        variable aux : std_logic := '0';
        begin
            nand1 <= not(j and clock and qb) after 2 ns;
            nand2 <= not(not(j) and clock and q) after 2 ns;
            q <= not(preset and qb and nand1) after 2 ns;
            qb <= not(clear and q and nand2) after 2 ns;

```




```

end process;
end flip_flop_d;

```

FLIP-FLOP D c/ CLEAR E PRESET

```

use work.util.all;
use work.std_logic_1164.all;

```

***** ENTIDADE FLIP-FLOP D C/ CP *****

```

entity flip_flop_cp is
port(j      : in std_logic := '0';
     k      : in std_logic := '1';
     preset  : in std_logic := '1';
     clear   : in std_logic := '1';
     clock   : in std_logic := '0';
     q       : inout std_logic := '0';
     qb      : inout std_logic := '1');
end flip_flop_cp;

```

***** ARQUITETURA DO FLIP-FLOP C/ CP *****

```

architecture flip_flop_cp of flip_flop_cp is
signal nand1,nand2 : std_logic := '1';

begin
  process (clock,preset,clear,nand1,nand2,q,qb)
    variable aux : std_logic := '0';
    begin
      nand1 <= not(j and clock and qb) after 2 ns;
      nand2 <= not(k and clock and q) after 2 ns;
      q    <= not(preset and qb and nand1) after 2 ns;
      qb   <= not(clear and q and nand2) after 2 ns;
    end process;
end flip_flop_cp;

```

REGISTRADOR PARALELO/SERIAL

```

use work.util.all;
use work.std_logic_1164.all;

```

***** ENTIDADE REG PARALELO/SERIAL *****

```

entity reg_ps is
port(carga    : in std_logic := '1';           -- habilitacao para a carga de dados

```




```

transfere : in std_logic := '0'; -- relógio para o descolamento dos dados
palavra : in dados := "00000000"; -- dados de entrada (array)
saida : inout std_logic := '0'; -- saída serial

```

```

end reg_ps;

```

***** ARQUITETURA DO REGISTRADOR P/S *****

```

architecture reg_ps of reg_ps is
    signal aux0,aux1,aux2,aux3,aux4,aux5,aux6,aux7,aux8 : std_logic := '0';
    signal inv : std_logic := '0';
    signal link0a,link0b,link1a,link1b,link2a,link2b,link3a,link3b : std_logic := '1';
    signal link4a,link4b,link5a,link5b,link6a,link6b,link7a,link7b : std_logic := '1';
    signal link8a,link8b,saidab : std_logic := '1';
    signal liga : std_logic := '0';

```

```

component flip_flop_cp
    port(j,k : in std_logic;
         preset : in std_logic;
         clear : in std_logic;
         clock : in std_logic;
         q : inout std_logic;
         qb : inout std_logic);
end component;

```

```

begin

```

```

    inv <= not(not(not(not(not(carga)))))) after 5 ns; -- habilita entrada de dados
    aux0 <= palavra(0) nand inv after 2 ns;
    aux1 <= palavra(1) nand inv after 2 ns;
    aux2 <= palavra(2) nand inv after 2 ns;
    aux3 <= palavra(3) nand inv after 2 ns;
    aux4 <= palavra(4) nand inv after 2 ns;
    aux5 <= palavra(5) nand inv after 2 ns;
    aux6 <= palavra(6) nand inv after 2 ns;
    aux7 <= palavra(7) nand inv after 2 ns;
    aux8 <= '1' nand inv after 2 ns;

```

```

flip8 : flip_flop_cp
    port map(j => link7a, k => link7b, preset => aux8, clear => carga, clock => transfere, q
=> saida, qb => saidab);

```

```

flip7 : flip_flop_cp
    port map(j => link6a, k => link6b, preset => aux7, clear => carga, clock => transfere, q
=> link7a, qb => link7b);

```

```

flip6 : flip_flop_cp
    port map(j => link5a, k => link5b, preset => aux6, clear => carga, clock => transfere, q
=> link6a, qb => link6b);

```

```

flip5 : flip_flop_cp

```



```
port map(j => link4a, k => link4b, preset => aux5, clear => carga, clock => transfere, q
=> link5a, qb => link5b);
```

```
flip4 : flip_flop_cp
port map(j => link3a, k => link3b, preset => aux4, clear => carga, clock => transfere, q
=> link4a, qb => link4b);
```

```
flip3 : flip_flop_cp
port map(j => link2a, k => link2b, preset => aux3, clear => carga, clock => transfere, q
=> link3a, qb => link3b);
```

```
flip2 : flip_flop_cp
port map(j => link1a, k => link1b, preset => aux2, clear => carga, clock => transfere, q
=> link2a, qb => link2b);
```

```
flip1 : flip_flop_cp
port map(j => link0a, k => link0b, preset => aux1, clear => carga, clock => transfere, q
=> link1a, qb => link1b);
```

```
flip0 : flip_flop_cp
port map(j => saida, k => saidab, preset => aux0, clear => carga, clock => transfere, q
=> link0a, qb => link0b);
```

```
end reg_ps;
```

REGISTRADOR SERIAL/PARALELO

```
use work.util.all;
use work.std_logic_1164.all;
```

***** ENTIDADE REG SERIAL/PARALELO *****

```
entity reg_sp is
port(input      : in std_logic := '0';           -- entrada de dados serial
      transfere  : in std_logic := '0';           -- relógio
      flag       : inout std_logic := '0';         -- sinalizador de saída dos dados
      palavra    : out dados1 := "000000000";     -- dados de entrada
      limpa      : in std_logic := '0');           -- limpa rsp
```

```
end reg_sp;
```

***** ARQUITETURA DO REGISTRADOR S/P *****

```
architecture reg_sp of reg_sp is
signal inv : std_logic := '0';
signal link0a,link0b,link1a,link1b,link2a,link2b,link3a,link3b : std_logic := '0';
signal link4a,link4b,link5a,link5b,link6a,link6b,link7a,link7b : std_logic := '0';
signal link8a,link8b : std_logic := '1';
```




```

component flip_flop_d
port(j      : in std_logic;
  clear    : in std_logic;
  clock    : in std_logic;
  q        : inout std_logic;
  qb       : inout std_logic);
end component;

begin

  inv <= not(not(not(not(not(limpa)))))) after 5 ns;
  flip1 : flip_flop_d
  port map(j => input, clear => inv, clock => transfere, q => link0a);

  flip2 : flip_flop_d
  port map(j => link0a, clear => inv, clock => transfere, q => link1a);

  flip3 : flip_flop_d
  port map(j => link1a, clear => inv, clock => transfere, q => link2a);

  flip4 : flip_flop_d
  port map(j => link2a, clear => inv, clock => transfere, q => link3a);

  flip5 : flip_flop_d
  port map(j => link3a, clear => inv, clock => transfere, q => link4a);

  flip6 : flip_flop_d
  port map(j => link4a, clear => inv, clock => transfere, q => link5a);

  flip7 : flip_flop_d
  port map(j => link5a, clear => inv, clock => transfere, q => link6a);

  flip8 : flip_flop_d
  port map(j => link6a, clear => inv, clock => transfere, q => link7a);

  flip9 : flip_flop_d
  port map(j => link7a, clear => inv, clock => transfere, q => flag);

  palavra(0) <= flag and link0a after 1 ns;
  palavra(1) <= flag and link1a after 1 ns;
  palavra(2) <= flag and link2a after 1 ns;
  palavra(3) <= flag and link3a after 1 ns;
  palavra(4) <= flag and link4a after 1 ns;
  palavra(5) <= flag and link5a after 1 ns;
  palavra(6) <= flag and link6a after 1 ns;
  palavra(7) <= flag and link7a after 1 ns;
  palavra(8) <= flag and flag after 1 ns;
end reg_sp;

```



COMPONENTES DA CHAVE QUADRADA

MEMÓRIA (tres flip-flops)

```
use work.util.all;
use work.std_logic_1164.all;
```

***** ENTIDADE MEMORIA *****

```
entity memoria is
port(bit1,bit2,bit3      : in std_logic := '0';
     habilita            : in std_logic := '0';
     bit_s1, bit_s2, bit_s3 : inout std_logic := '0');
end memoria;
```

***** ARQUITETURA DA MEMORIA *****

```
architecture memoria of memoria is
begin
    process (bit1,bit2,bit3,habilita)
```

bit1 - habilitacao dos canais
bit2 - habilitacao dos canais
bit3 - habilitacao dos canais
habilita - controle para a entrada de informacoes na chave

```
variable nand1,nand2,bit_s11 : std_logic;
variable nand3,nand4,bit_s22 : std_logic;
variable nand5,nand6,bit_s33 : std_logic;
begin
```

flip-flop1

```
nand1 := bit1 nand habilita;
nand2 := not(bit1) nand habilita;
bit_s11 := bit_s1 nand nand2;
bit_s1 <= bit_s11 nand nand1 after 4 ns;
```

flip-flop2

```
nand3 := bit2 nand habilita;
nand4 := not(bit2) nand habilita;
bit_s22 := bit_s2 nand nand4;
bit_s2 <= bit_s22 nand nand3 after 4 ns;
```


flip-flop3

```
nand5 := bit3 nand habilita;
nand6 := not(bit3) nand habilita;
bit_s33 := bit_s3 nand nand6;
bit_s3 <= bit_s33 nand nand5 after 4 ns;

end process;
end memoria;
```

DECODIFICADOR (GERA DADOS PARA O MUX/DEMUX)

```
use work.util.all;
use work.std_logic_1164.all;
```

***** ENTIDADE DECODIFICADOR (VEM DA MEMORIA) *****

```
entity decodificador2 is
port(mem1,mem2,mem3      : in std_logic := '0';
      mult1,mult2,mult3,mult4 : out std_logic);
```

```
mem1,mem2,mem3 - codigos vindo da memoria
mult1,mult2,mult3,mult4 - codigos decodificados
```

```
end decodificador2;
```

***** ARQUITETURA DO DECODIFICADOR2 *****

```
architecture decodificador2 of decodificador2 is
begin
  process (mem1,mem2,mem3)
  begin
    mult1 <= (mem1 and mem2 and not(mem3)) after 4 ns;
    mult2 <= (mem1 and not(mem2)) after 4 ns;
    mult3 <= (mem1 and not(mem2)) or (not(mem1) and mem2) or (mem2 and
not(mem3)) after 7 ns;
    mult4 <= ((not(mem2) and mem3) or (not(mem1) and mem3) or (mem1 and
mem2 and not(mem3))) after 7 ns;
  end process;
end decodificador2;
```

MULTIPLEXADOR

```
use work.util.all;
```



```
use work.std_logic_1164.all;
```

******* ENTIDADE MULTIPLEXADOR (VEM DO DECODIFICADOR) *******

```
entity mux is
port(a,b          : in std_logic := '0';
     ent11,ent12,ent13 : in std_logic := '0';
     ent21,ent22,ent23 : in std_logic := '0';
     ent31,ent32,ent33 : in std_logic := '0';
     ent41,ent42,ent43 : in std_logic := '0';
     s1,s2,s3        : out std_logic := '0');
```

a,b - controle do multiplexador
ent11,ent12,..., ent43 - entrada de dados para o mux
s1,s2,s3 - saída de dados do mux

```
end mux;
```

******* ARQUITETURA DO MULTIPLEXADOR *******

```
architecture mux of mux is
```

```
begin
```

```
process(ent11,ent12,ent13,ent21,ent22,ent23,ent31,ent32,ent33,ent41,ent42,ent43)
variable and1,and2,and3,and4 : std_logic := '0';
```

```
begin
```

```
and1 := not(a) and not(b);
```

```
and2 := not(a) and b;
```

```
and3 := a and not(b);
```

```
and4 := a and b;
```

```
s1 <= (and1 and ent11) or (and2 and ent21) or (and3 and ent31) or (and4 and ent41) after 6 ns;
```

```
s2 <= (and1 and ent12) or (and2 and ent22) or (and3 and ent32) or (and4 and ent42) after 6 ns;
```

```
s3 <= (and1 and ent13) or (and2 and ent23) or (and3 and ent33) or (and4 and ent43) after 6 ns;
```

```
end process;
```

```
end mux;
```

DEMULTIPLEXADOR

```
use work.util.all;
use work.std_logic_1164.all;
```



******* ENTIDADE DEMULTIPLEXADOR *******

```
entity demux is
port(c,d                      : in std_logic := '0';
     entr1,entr2,entr3        : in std_logic := '0';
     saida11, saida12, saida13 : out std_logic;
     saida21, saida22, saida23 : out std_logic;
     saida31, saida32, saida33 : out std_logic;
     saida41, saida42, saida43 : out std_logic);
```

c,d - programacao do demultiplexador
entr1,entr2,entr3 - entrada para o demux
saida11,saida12,..., saida43 - saidas do demux

end demux;

******* ARQUITETURA DO DEMULTIPLEXADOR *******

architecture demux of demux is

```
begin
  process(entr1,entr2,entr3)
    variable and5,and6,and7,and8 : std_logic := '0';
    begin
      and5 := not(c) and not(d);
      and6 := not(c) and d;
      and7 := c and not(d);
      and8 := c and d;
```

PRIMEIRO CANAL DE SAIDA

```
saida11 <= (entr1 and and5) after 7 ns;
saida12 <= entr2 and and5 after 7 ns;
saida13 <= entr3 and and5 after 7 ns;
```

SEGUNDO CANAL DE SAIDA

```
saida21 <= entr1 and and6 after 7 ns;
saida22 <= entr2 and and6 after 7 ns;
saida23 <= entr3 and and6 after 7 ns;
```

TERCEIRO CANAL DE SAIDA

```
saida31 <= entr1 and and7 after 7 ns;
saida32 <= entr2 and and7 after 7 ns;
saida33 <= entr3 and and7 after 7 ns;
```

QUARTO CANAL DE SAIDA

```
saida41 <= entr1 and and8 after 6 ns;
```




```
saida42 <= entr2 and and8 after 6 ns;
saida43 <= entr3 and and8 after 6 ns;
```

```
end process;
end demux;
```

CHAVE

```
use work.util.all;
use work.std_logic_1164.all;
```

***** ENTIDADE DA CHAVE *****

```
entity chave is
port(memoria1,memoria2,memoria3 : in std_logic := '0';
     enable                      : in std_logic := '0';
     e11,e12,e13                : in std_logic := '0';
     e21,e22,e23                : in std_logic := '0';
     e31,e32,e33                : in std_logic := '0';
     e41,e42,e43                : in std_logic := '0';
     s11,s12,s13                : out std_logic;
     s21,s22,s23                : out std_logic;
     s31,s32,s33                : out std_logic;
     s41,s42,s43                : out std_logic);
```

memoria1,memoria2,memoria3 - codigo de enderecamento dos canais
e11,e12, ..., e33 - canais de entrada na chave
s11,s12, ..., s33 - canais de saida da chave

```
end chave;
```

***** Arquitetura da CHAVE *****

```
architecture chave of chave is
```

```
component mux
```

```
port (a,b,ent11,ent12,ent13,ent21,ent22,ent23,ent31,ent32,ent33,ent41,ent42,ent43
      : in std_logic;
      s1,s2,s3      : out std_logic);
end component;
```

```
component demux
```

```
port (c,d          : in std_logic;
      entr1,entr2,entr3 : in std_logic;
      saida11,saida12,saida13,saida21,saida22,saida23,saida31,saida32,saida33,
      saida41,saida42,saida43 : out std_logic);
end component;
```

```
component decodificador2
```

```
port(mem1,mem2,mem3      : in std_logic;
```




```

        mult1,mult2,mult3,mult4    : out std_logic);
end component;

```

```

component memoria
    port(bit1,bit2,bit3          : in std_logic;
          habilita                : in std_logic;
          bit_s1, bit_s2, bit_s3  : inout std_logic);
end component;

```

```

signal cond1a,cond2a,cond3a : std_logic := '0';    -- conduz os dados de entrada
signal cond1b,cond2b,cond3b : std_logic := '0';    -- conduz os dados de saída
signal sel1,sel2,sel3,sel4   : std_logic := '0';    -- conduz os dados que saem do
decodificador
signal smem1,smem2,smem3 : std_logic :='0';        -- canais de conexao para a
memoria

```

saídas secundárias dos demultiplexadores

```

signal sa11,sa12,sa13      : std_logic;
signal sa21,sa22,sa23      : std_logic;
signal sa31,sa32,sa33      : std_logic;
signal sa41,sa42,sa43      : std_logic;
signal sb11,sb12,sb13      : std_logic;
signal sb21,sb22,sb23      : std_logic;
signal sb31,sb32,sb33      : std_logic;
signal sb41,sb42,sb43      : std_logic;

```

```

begin

```

```

    s11 <= sa11 or sb11 after 2 ns;
    s12 <= sa12 or sb12 after 2 ns;
    s13 <= sa13 or sb13 after 2 ns;
    s21 <= sa21 or sb21 after 2 ns;
    s22 <= sa22 or sb22 after 2 ns;
    s23 <= sa23 or sb23 after 2 ns;
    s31 <= sa31 or sb31 after 2 ns;
    s32 <= sa32 or sb32 after 2 ns;
    s33 <= sa33 or sb33 after 2 ns;
    s41 <= sa41 or sb41 after 2 ns;
    s42 <= sa42 or sb42 after 2 ns;
    s43 <= sa43 or sb43 after 2 ns;

```

MUX/DEMUX

```

    m_B : mux

```

```

        port map(a => sel3, b => sel4, s1 => cond1b, s2 => cond2b, s3 => cond3b, ent11
=> e11, ent12 => e12, ent13 => e13, ent21 => e21, ent22 => e22, ent23 => e23,
ent31 => e31, ent32 => e32, ent33 => e33, ent41 => e41, ent42 => e42, ent43 =>
e43);

```

```

    d_B : demux

```

```

        port map(c => sel1, d => sel2, entr1 => cond1b, entr2 => cond2b, entr3 =>
cond3b, saida11 => sb11, saida12 => sb12, saida13 => sb13, saida21 => sb21,
saida22 => sb22, saida23 => sb23, saida31 => sb31, saida32 => sb32, saida33 =>
sb33, saida41 => sb41, saida42 => sb42, saida43 => sb43);

```



MUX/DEMUX

```
m_A : mux
  port map(a => sel1, b => sel2, s1 => cond1a, s2 => cond2a, s3 => cond3a,
ent11 => e11, ent12 => e12, ent13 => e13, ent21 => e21, ent22 => e22, ent23 =>
e23, ent31 => e31, ent32 => e32, ent33 => e33, ent41 => e41, ent42 => e42, ent43
=> e43);
d_A : demux
  port map(c => sel3, d => sel4, entr1 => cond1a, entr2 => cond2a, entr3 =>
cond3a, saida11 => sa11, saida12 => sa12, saida13 => sa13, saida21 => sa21,
saida22 => sa22, saida23 => sa23, saida31 => sa31, saida32 => sa32, saida33 =>
sa33, saida41 => sa41, saida42 => sa42, saida43 => sa43);
```

DECODIFICADOR DAS CHAVES

```
decod : decodificador2
  port map(mem1 => smem1, mem2 => smem2, mem3 => smem3, mult1 => sel1,
mult2 => sel2, mult3 => sel3, mult4 => sel4);
```

MEMORIA DAS CHAVES

```
memory : memoria
  port map(bit1 => memoria1, bit2 => memoria2, bit3 => memoria3, habilita =>
enable, bit_s1 => smem1, bit_s2 => smem2, bit_s3 => smem3);
end chave;
```

INTERLIGAÇÃO DOS MODULOS

```
use work.util.all;
use work.std_logic_1164.all;
```

******* ENTIDADE DO TOTAL *******

```
entity total is
port(busNE1,busNE2,busNE3      : inout std_logic := '0'; -- canais de entrada norte
busNS1,busNS2,busNS3          : inout std_logic := '0'; -- canais de saida norte
busSE1,busSE2,busSE3          : inout std_logic := '0'; -- canais de entrada sul
busSS1,busSS2,busSS3          : inout std_logic := '0'; -- canais de saida sul
busLE1,busLE2,busLE3          : inout std_logic := '0'; -- canais de entrada leste
busLS1,busLS2,busLS3          : inout std_logic := '0'; -- canais de saida oeste
busOE1,busOE2,busOE3          : inout std_logic := '0'; -- canais de entrada oeste
busOS1,busOS2,busOS3          : inout std_logic := '0'; -- canais de saida oeste
cont2,cont3,cont4              : in std_logic := '0'; -- selecao do canal de comunicacao
cont1                          : in std_logic := '0'; -- habilita a chave para a conexao de proc.
```



```

wordN      : in dados := "00000000"; -- palavra de entrada no proc. norte
wordS      : in dados := "00000000"; -- palavra de entrada no proc. sul
wordL      : in dados := "00000000"; -- palavra de entrada no proc. leste
wordO      : in dados := "00000000"; --palavra de entrada no proc. oeste
wordN_cheg : out dados := "00000000"; --palavra de chegada no norte
wordS_cheg : out dados := "00000000"; -- palavra de chegada no sul
wordL_cheg : out dados := "00000000"; -- palavra de chegada no leste
wordO_cheg : out dados := "00000000"); -- palavra de chegada no oeste

```

end total;

***** ARQUITETURA DO TOTAL *****

architecture total of total is

component reg_ps

```

port(carga      : in std_logic;
      transfere  : in std_logic;
      palavra    : in dados;
      saida      : inout std_logic);

```

end component;

component processador

```

port(inf_part   : in dados;
      inf_cheg   : in dados1;
      registrador1 : inout dados;
      registrador2 : out dados;
      hab_rps    : out std_logic;
      pulso      : out std_logic;
      rsp_rem    : in std_logic;
      rsp_local  : inout std_logic);

```

end component;

component reg_sp

```

port(input      : in std_logic;
      transfere  : in std_logic;
      palavra    : out dados1;
      flag       : inout std_logic;
      limpa      : in std_logic);

```

end component;

component chave

```

port(memoria1,memoria2,memoria3 : in std_logic;
      enable                     : in std_logic;
      e11,e12,e13                : in std_logic := '0';
      e21,e22,e23                : in std_logic := '0';
      e31,e32,e33                : in std_logic := '0';
      e41,e42,e43                : in std_logic := '0';
      s11,s12,s13                : out std_logic;
      s21,s22,s23                : out std_logic;
      s31,s32,s33                : out std_logic;
      s41,s42,s43                : out std_logic);

```



end component;

```
signal linkN_rps,linkS_rps : std_logic := '1'; -- canais para o sinal de carga do rps
signal linkL_rps,linkO_rps : std_logic := '1'; -- canais para o sinal de carga do rps
signal limpaN,limpaS : std_logic := '0'; -- limpa rsp
signal limpaL,limpaO : std_logic := '0'; -- limpa rsp
```

```
signal infN_part : dados := "000000000"; -- informacao de partida do proc. norte
signal infS_part : dados := "000000000"; -- informacao de partida do proc. sul
signal infL_part : dados := "000000000"; -- informacao de partida do proc. leste
signal infO_part : dados := "000000000"; -- informacao de partida do proc. oeste
```

```
signal infN_cheg : dados1 := "000000000"; -- informacao de chegada no proc. norte
signal infS_cheg : dados1 := "000000000"; -- informacao de chegada no proc. sul
signal infL_cheg : dados1 := "000000000"; -- informacao de chegada no proc. leste
signal infO_cheg : dados1 := "000000000"; -- informacao de chegada no proc. oeste
```

begin

PROCESSADOR N

```
rspN : reg_sp
port map(input => busNS1, transfere => busNS2, flag => busNE3, limpa => limpaN,
palavra => infN_cheg);
```

```
rpsN : reg_ps
port map(carga => linkN_rps, transfere => busNE2, saida => busNE1, palavra =>
infN_part);
```

```
processadorN : processador
port map(inf_part => wordN, registrador1 => infN_part, hab_rps => linkN_rps, pulso
=> busNE2, rsp_rem => busNS3, rsp_local => limpaN, inf_cheg => infN_cheg,
registrador2 => wordN_cheg);
```

PROCESSADOR S

```
rspS : reg_sp
port map(input => busSS1, transfere => busSS2, flag => busSE3, limpa => limpaS,
palavra => infS_cheg);
```

```
rpsS : reg_ps
port map(carga => linkS_rps, transfere => busSE2, saida => busSE1, palavra =>
infS_part);
```

```
processadorS : processador
port map(inf_part => wordS, registrador1 => infS_part, hab_rps => linkS_rps, pulso
=> busSE2, rsp_rem => busSS3, rsp_local => limpaS, inf_cheg => infS_cheg,
registrador2 => wordS_cheg);
```



PROCESSADOR L

```
rspL : reg_sp  
port map(input => busLS1, transfere => busLS2, flag => busLE3, limpa => limpaL,  
palavra => infL_cheg);
```

```
rpsL : reg_ps  
port map(carga => linkL_rps, transfere => busLE2, saida => busLE1, palavra =>  
infL_part);
```

```
processadorL : processador  
port map(inf_part => wordL, registrador1 => infL_part, hab_rps => linkL_rps, pulso =>  
busLE2, rsp_rem => busLS3, rsp_local => limpaL, inf_cheg => infL_cheg,  
registrador2 => wordL_cheg);
```

PROCESSADOR O

```
rspO : reg_sp  
port map(input => busOS1, transfere => busOS2, flag => busOE3, limpa => limpaO,  
palavra => infO_cheg);
```

```
rpsO : reg_ps  
port map(carga => linkO_rps, transfere => busOE2, saida => busOE1, palavra =>  
wordO);
```

```
processadorO : processador  
port map(inf_part => wordO, registrador1 => infO_part, hab_rps => linkO_rps, pulso  
=> busOE2, rsp_rem => busOS3, rsp_local => limpaO, inf_cheg => infO_cheg,  
registrador2 => wordO_cheg);
```

CONEXAO DOS CANAIS VINDOS DOS PROC. COM A CHAVE

```
CONEXAO : chave  
port map(enable => cont1, memoria1 => cont2, memoria2 => cont3, memoria3 =>  
cont4, e11 => busNE1, e12 => busNE2, e13 => busNE3, e21 => busSE1, e22 =>  
busSE2, e23 => busSE3, e31 => busLE1, e32 => busLE2, e33 => busLE3, e41 =>  
busOE1, e42 => busOE2, e43 => busOE3, S11 => busNS1, s12 => busNS2, s13 =>  
busNS3, s21 => busSS1, s22 => busSS2, s23 => busSS3, s31 => busLS1, s32 =>  
busLS2, s33 => busLS3, s41 => busOS1, s42 => busOS2, s43 => busOS3);
```

```
end total;
```


2 - Análise Temporal do Sistema

Nesta parte da dissertação faz-se uma análise dos atrasos envolvidos no tráfego da informação pelo sistema. Para esta análise, foi utilizado o simulador VSIM da *Model Technology 1.3*.

Um atraso é o intervalo de tempo que um sinal elétrico leva para atravessar um componente do circuito. A existência dos atrasos se deve as propriedades físico-químicas dos elementos que compõem um circuito, as quais provocam certa resistência à movimentação dos elétrons.

Como o projeto das chaves visa o nível lógico, vai-se considerar como atraso o intervalo tempo necessário para a realização de operações lógicas sobre os dados de entrada.

Neste projeto, adota-se atrasos fixos para cada um dos tipos de portas lógicas. Por exemplo:

- **AND** - atraso de 3 ns;
- **OR** - atraso de 3 ns;
- **NAND** - atraso de 2 ns;
- **NOR** - atraso de 2 ns;
- **NOT** - atraso de 1 ns;

Na tabela 11, é exibido o tempo necessário para o tráfego da informação em cada um dos módulos que compõem a chave.

Instante de Tempo (ns)	AÇÃO
0	Controlador envia palavra de controle
1	Palavra de controle chega na chave
5	Palavra de controle está armazenada na memória
12	Decodificador finaliza processamento
13	Dados estão disponíveis nas entradas do multiplexador
28	Dados estão disponíveis nas saídas do demultiplexador

Tabela 11 - Tempo de Habilitação dos Componentes da Chave

Pelo exposto na tabela 11, pode-se concluir que:

- a chave só estará habilitada 12 ns após ter recebido a palavra de controle.
- os dados que passam pela chave sofrem atrasos de 15 ns.

Na tabela 12, é apresentado o comportamento temporal dos registradores serial/paralelo e paralelo/serial, durante o tráfego da informação.



Instante de tempo (ns)	Pulso de RPS	Reg. Par/Ser	Pulso de RSP	Reg. Ser/Par
1	0	Recebe dados do proc.	0	
12	0	Habilitada a 1ª saída	0	
13	1		0	
18	0		0	
19	0	Habilitada a 2ª saída	0	
20	1		0	
25	0		0	
26	0	Habilitada a 3ª saída	0	
27	1		0	Chega o 1º bit de dado
28	1		1	
32	0		1	
33	0	Habilitada a 4ª saída	0	
34	1		0	Chega o 2º bit de dado
35	1		1	
39	0		1	
40	0	Habilitada a 5ª saída	0	
41	1		0	Chega o 3º bit de dado
42	1		1	
46	0		1	
47	0	Habilitada a 6ª saída	0	
48	1		0	Chega o 4º bit de dado
49	1		1	
53	0		1	
54	0	Habilitada a 7ª saída	0	
55	1		0	Chega o 5º bit de dado
56	1		1	
60	0		1	
61	0	Habilitada a 8ª saída	0	
62	1		0	Chega o 6º bit de dado
63	1		1	
67	0		1	
68	0	Habilitada a 1ª saída	0	
69	1		0	Chega o 7º bit de dado
70	1		1	
74	0		1	
75	0	Habilitada a 2ª saída	0	
76	1		0	Chega o 8º bit de dado

Tabela 12 - Comportamento Temporal dos Registradores



Pelo exposto na tabela 12, pode-se observar que:

- os pulsos emitidos pelo processador que envia dados têm uma frequência de 7 ns.
- o registrador serial/paralelo só começa receber os *bits* de dados 15 ns após sua emissão.
- um novo *bit* de dado será transmitido pelo registrador paralelo/serial, 1 ns após o relógio ir ao nível 0.
- 1 ns após o relógio ir ao nível 0, um novo *bit* de dado chega no registrador serial/paralelo.
- após emitir o último *bit*, o registrador paralelo/serial volta a repetir a transmissão dos dados.
- o tempo médio para a transmissão de uma palavra de 9 *bits* é de 77 ns.

Conclusões e Perspectivas

Neste trabalho, realizou-se o projeto lógico de uma chave quadrada, responsável pela conexão de dois dos quatro canais ligados a ela. Adicionalmente, elaborou-se um protocolo de comunicação para os processadores do arranjo.

Utilizou-se a linguagem de descrição de *hardware* VHDL, para representar um sistema composto por 4 protocolos de comunicação e uma chave. Posteriormente, utilizou-se um simulador para a verificação de seu comportamento.

A ferramenta de modelagem VHDL mostrou-se adequada, dadas as facilidades oferecidas para a representação de estruturas complexas de sinais em estruturas simples e fáceis de manusear. Especificamente, a representação comportamental do protocolo de comunicação e a utilização de variáveis binárias para a representação dos canais de comunicação.

Para modelar o projeto em VHDL foram utilizados 2 níveis de abstração. A chave e os registradores foram representados em nível estrutural, permitindo verificar suas funcionalidades e os tempos de execução de cada um dos módulos que os compõem. Já, a lógica responsável pelo sincronismo do sistema, interna a cada processador, foi representada comportamentalmente, devido à sua complexidade.

Considerando as convenções de atrasos utilizadas para cada porta lógica, tem-se que o tempo médio de travessia de um dado pela chave é no mínimo de 15 ns.

Adicionalmente, adotando-se um palavra de dados de 8 *bits*, tem-se que o tempo médio que esta palavra leva para sair de um processador e chegar a outro é da ordem de 77 ns. Portanto, para que se evitem perdas de dados devem-se ter ciclos de cálculo nos processadores, próximos ao tempo necessário às suas transmissões pelo barramento.

Como sugestão para trabalhos futuros propõe-se o projeto estrutural da lógica que sincroniza a troca de informações entre os processadores e a implementação de um protótipo do sistema descrito nesta dissertação.

10. Marrangoni, T. "A nova Especificação de uma Arquitetura para a Simulação de Circuitos VLSI". Anais do VII Simpósio Brasileiro de Conceção de Circuitos Integrados, pp. 347-355, Gramado, 1994.

11. Marrangoni, T. "A nova Arquitetura para a Simulação de Circuitos VLSI". Trabalho de Conclusão de Curso (Doutorado) - Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, 1992.



Referências Bibliográficas

1. Allison, J. "Eletronic Integrated Circuits" McGraw-Hill Book Co. Ltd., 139pp., 1975
2. Armstrong, J. R. "Chip-level Modelling with VHDL" Prentice-Hall Inc., 148pp., 1989
3. Baranauskas, V. "Tecnologia dos Cristais de Silício em Microeletrônica" Editora da Unicamp, 155pp., 1990
4. Baranauskas, V. (editor) "Processos de Microeletrônica" R. Vieira Gráfica e Editora Ltda., 281pp., 1990
5. Clare "Desiging Logic Systems Using State Machine" McGraw-Hill Book Co., 1973.
6. Deboo, G. J. e Burros, C. N. "Integrated Circuits and Semiconductor Devices: Theory and Applications" - Appendix A 'Basic Semiconductor
7. Feltre, R. e Yoshinaga, S. "Físico-Química" Editora Moderna Ltda, pp. 126-133, 1968
8. Hill, F. J. e Peterson, G. R. "Computer-Aided Logical Design-with Emphasis in VLSI" capítulo 12 'VLSI Realization of Digital Systems' John Wiley & Sons Inc., pp. 379-386, 1993
9. Mano, M. M. "Digital Logic and Computer Design" Prentice-Hall, Inc. 1979
10. Marranghello e F. Damiani "Especificações de uma Arquitetura para a Simulação de Circuitos" Anais do VIII Simpósio Brasileiro de Concepção de Circuitos Integrados,, pp. 347-356, Gramado, 1994
11. Marranghello N. "Uma Metodologia para a Simulação de Circuitos ULSI". Campinas, 1992, Tese (Doutoramento) - Faculdade de Engenharia Elétrica. Universidade Estadual de Campinas, 1992.



12. Marranghello, N. "*Apostila de Métodos de Análise de Sistemas Digitais*". Curso de Bacharelado em Ciência da Computação, Universidade Estadual Paulista, São José do Rio Preto, 1995
13. Muroga S. "*Logic Design and Switching Theory*", Robert and Kriegen Publishing Corp, Malabar, Flórida, 1990.
14. Prather, R. E. "*Introduction to Switching Theory: A Mathematical Approach*", Allyn and Bacon, Inc., 1967.
15. Russel, G. Sayers, I. L. "*Advanced simulation and test methodologies for VLSI design*" - Chapter 3 'simulation techniques for empirical design verification' - Van Nostrand Reinhold, 1989
16. Sedra, A. S. e Smith, K. C. "*Microeletronic Circuits*" Saunders College Publishing, 1157 pp., 1991
17. Taub, H. "*Circuitos digitais e microprocessadores*" McGraw-Hill Corp, São Paulo, 1984
18. W. Machaca Luque "*Modelamento em VHDL de um sistema de simulação de circuitos*" dissertação de mestrado, FEE/UNICAMP, 1994
19. Wagner F. R., Porto I. J., Weber R. F. e Weber T. S. "*Métodos de Validação de Sistemas Digitais*" VI Escola de Computação, Campinas, SP, 199pp., 1988
20. Weste, N. e Eslivaghian, K. "*Principles of CMOS VLSI Design*" Addison-Wesley, Inc., 531pp., 1985



